Complex & Intelligent Systems (2021) 7:3135–3163
https://doi.org/10.1007/s40747-021-00507-6

ORIGINAL ARTICLE

Hyper-heuristic approach: automatically designing adaptive mutation operators for evolutionary programming

Libin Hong[1] · John R. Woodward[2] · Ender Özcan[3] · Fuchang Liu[1]

Received: 7 September 2020 / Accepted: 12 August 2021 / Published online: 28 August 2021

Abstract
Genetic programming (GP) automatically designs programs. Evolutionary programming (EP) is a real-valued global optimisation method. EP uses a probability distribution as a mutation operator, such as Gaussian, Cauchy, or Lévy distribution. This study proposes a hyper-heuristic approach that employs GP to automatically design different mutation operators for EP. At each generation, the EP algorithm can adaptively explore the search space according to historical information. The experimental results demonstrate that the EP with adaptive mutation operators, designed by the proposed hyper-heuristics, exhibits improved performance over other EP versions (both manually and automatically designed). Many researchers in evolutionary computation advocate adaptive search operators (which do adapt over time) over non-adaptive operators (which do not alter over time). The core motive of this study is that we can automatically design adaptive mutation operators that outperform automatically designed non-adaptive mutation operators.

Keywords Hyper-heuristic · Evolutionary programming · Genetic programming · Adaptive mutation

Introduction

Genetic programming (GP) [1] is a branch of evolutionary computation that can generate computer programs, and is widely applied in numerous fields [2–10]. Evolutionary programming (EP) is a black-box optimiser and mutation is the only operator in EP. Researchers recommended different probability distributions as mutation operators and analyse their characteristics. For example, Yao et al. [11] point out that a Cauchy mutation performs better than a Gaussian mutation by virtue of a higher probability of making large jumps, while large step sizes are typically detrimental towards the end of the search process when the set of current search points are close to the global optimum. Hong et al. [12] mentioned that when using a non-adaptive mutation operator in EP, more offspring usually survive in the early generations, and conversely less survive in the later generations of the run. Researchers also proposed different mutation strategies to promote EP efficiency [12–17]. 'A hyper-heuristic is a search method or learning mechanism for selecting or generating heuristics to solve computational search problems' [18]. Researchers classify hyper-heuristics according to the feedback sources in the learning process: Online learning hyper-heuristics learn from a single instance of a problem; Offline learning hyper-heuristics learn from a set of training instances and generalise to unseen instances [18]. Both online [2–5] and offline hyper-heuristics [6–10] are applied to various research fields.

Hyper-heuristics are an effective and popular technique which have been applied to a wide range of problem domains. Cowling et al. [2] used online learning hyper-heuristics to minimise the number of delegates who actually attend the sales summit out of a number of possible delegate attendees. Dowsland et al. [3] used online learning hyper-heuristics to handle the design and evaluation of a heuristic solution to the problem of selecting a set of shippers that minimises the total annual volume of space required to accommodate a given set of products with known annual shipment quantities. Ochoa et al. [4] used online learning hyper-heuristics to describe the number of extensions to the HyFlex framework that enables the implementation of more robust and effective adaptive search heuristics. Pisinger et al. [5] used online learning hyper-heuristics to present a unified heuristic which is able to

✉ Fuchang Liu
liufc@hznu.edu.cn

Libin Hong
libin.hong@hznu.edu.cn

1 Hangzhou Normal University, Hangzhou, China

2 Queen Mary University of London, London, UK

3 University of Nottingham, Nottingham, UK

solve five different variants of the vehicle routing problem. Shao et al. [6] used multiobjective genetic programming as an offline learning hyper-heuristic to apply the feature learning for image classification. Hong et al. [7,8] used GP as an offline learning hyper-heuristic to automatically design a mutation operator for EP and automatically design more general mutation operators for EP [9]. Ross et al. [10] used an offline learning hyper-heuristic to represent a step towards a new method of using evolutionary algorithms that may solve some problems of acceptability for real-world use.

In this study, the heuristics are *adaptive* mutation operators generated by GP based on an offline hyper-heuristic. In other words, we use a GP-based offline hyper-heuristic to redesign a portion of the EP algorithm (i.e. the probability distribution) to improve the overall EP algorithm performance. The contribution of this study is that this work realises the 'automatic' design of 'adaptive' mutation operators, it realises both 'automatic' and 'adaptive' at the same time has been revised and updated. Previous studies either automatically designed static/non-adaptive mutation operators [7–9], or human/manually designed adaptive mutation operators/mutation strategies for EP [13,16,17,19–22]. To achieve this target, a set of adaptive factors was proposed, which collected and updated historical information during the evolutionary process. The proposed method also contributes to a group of automatically designed adaptive mutation operators for function classes. In essence, these adaptive factors are variables and are provided in terminal sets for GP that uses these variables to automatically design a mutation operator (random number generator), which replaces the human designed mutation operator in the EP algorithm. In EP, each individual is taken as a pair of real-valued vectors. The variables can partly reflect the individuals' current position and the evolutionary status. These variables change during evolution, and affect the characteristics of mutation, thus we call them adaptive factors. For example, CUR_MIN_X is the minimum value of the best individuals up to the current generation. $N(\mu, CUR\_MIN\_X)$ is a mutation operator, CUR_MIN_X is updated in each EP generation, then the size of the jumps the mutation operator can make keeps updating in each EP generation. CUR_MIN_X is an adaptive factor type. The adaptive factors are collected and employed for EP in both the training and testing stages.

The hypothesis of this study, inspired by [8,9,11,12] is that for a specific class of functions GP can design an *adaptive* mutation operator for EP, which outperforms an automatically designed non-adaptive mutation operator. The adaptive factors collected through the EP run can lead to the size of the jumps being different. A set of *adaptive* mutation operators can be discovered for function classes respectively. For more details please refer to Sect. 4.

The outline of this paper is as follows: in Sect. 2, we describe function optimisation and the basic EP algorithm. In Sect. 3, we describe the adaptive mutation operator. Section 4 describes connections between GP and EP. Section 5 describes the benchmark functions and their classes, and how a mutation operator is trained, and also describes the test results. Here, we contrast the performance of previously proposed automatically designed non-adaptive mutation operators (ADMs, i.e. not adaptive) with automatically designed adaptive mutation operators (ADAMs, i.e. are adaptive). In Sect. 6, we analyse and compare the testing results and in Sect. 7, we summarise and conclude the paper.

The basic EP algorithm

The EP algorithm evolves a population of numerical vectors to find near-optimum functional values. Mutation is the only operator for EP, and recently EP researchers have focussed primarily on manually designing mutation operators or smart strategies to use the mutation operators [11–16,23].

Minimisation can be formalised 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. S is the problem search space (function f). The aim of EP is to find a point $x_{\min} \in S$ such that $f(x_{\min})$ is a global minimum on S or a close approximation. More specifically, the requirement is to find an $x_{\min} \in S$ such that

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

f does not need to be continuous or differentiable, but it must be bounded (i.e. S is bounded). The EP's mutation process is represented by the following equations:

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

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

In the above equations, each individual is taken as a pair of real-valued vectors, $(x_i, \eta_i), \forall i \in \{1, \cdots, \mu\}$, μ represents number of individuals in the population, i is the dimensionality of f, and j represents the $j-th$ component of the vectors x_i, x_i', η_i, and η_i', the factors γ and γ' are set to $(\sqrt{2\sqrt{n}})^{-1}$ and $(\sqrt{2n})^{-1}$. D_j represents the mutation operator; researchers usually use a Cauchy, Gaussian, or Lévy distribution $L_{\alpha,\gamma}(y)$ as the mutation operator [11,14,23]. Lee et al. [14] point out that the Lévy distribution with $\alpha = 1.0$ is the Cauchy distribution, and that with $\alpha = 2.0$, it is the Gaussian distribution. For a complete EP description, refer to [24].

In this study, the hyper-heuristic framework designs an adaptive mutation operator (can also be seen as an adaptive mutation strategy) and replaces a probability distribution D_j. The EP algorithm uses this candidate mutation operator on functions generated from the function classes.

Adaptive mutation operator

This study focuses on automatically designing adaptive heuristics (i.e. random number generators which are used as mutation operators in EP). However, the human designed adaptive heuristics have already been proposed in numerous studies, below are examples of human designed heuristics that relate to adaptiveness, adaptive mutation operator, and mutation strategies:

The concept of adaptiveness is widely used in many technologies: in [25] they proposed new local search procedures combined in an adaptive large neighbourhood search metaheuristic to generate solutions for the gate matrix layout problem. An adaptive charged system search algorithm was developed to solve economic dispatch problems in [26]. To make the terminal phase of the standard global positioning system and inertial navigation system (GPS/INS) landing system more precise, an adaptive fuzzy data fusion algorithm was developed yielding more accurate state estimates while the vehicle approaches the landing surface [27].

A novel adaptive strategy is developed to dynamically tune the control parameters of the random learning operator so that the improved adaptive human learning optimisation can efficiently accelerate the convergence at the beginning of an iteration, develop diversity in the middle of the searching process to better explore the solution space, and perform an accurate local search at the end of the search to find the optima [28].

A hybrid adaptive evolutionary algorithm was introduced to improve the performance of the search operators across the various stages of the search/optimisation process of evolutionary algorithms [29]. A neighbourhood-adaptive differential evolution method was also proposed; in this framework, multiple neighbourhood relationships are defined for each individual and the neighbourhood relationships are then adaptively selected for specific functions during the evolutionary process [30].

An adaptive switching particle swarm optimisation algorithm using a hybrid update sequence is proposed, which can automatically switch to synchronous or asynchronous updating during the evolutionary process [31]. [32] presented a method for reusing the valuable information available from previous individuals to guide later search; in this approach, prior useful information was fed back to the updating process.

The concept of adaptive mutation operators has been proposed in numerous studies. The adaptive mutation operator proposed for particle swarm optimisation uses the three mutation operators (Cauchy, Gaussian, and Lévy) in [33]; the mutation operators that cause lower fitness values for the offspring see their selection ratios decreased, and the selection ratios for mutation operators that cause higher fitness values for the offspring increase. In [34], they developed an adaptive mutation for a particle swarm optimisation for an airfoil aerodynamic design. In [35] they proposed using an adaptive strategy in differential evolution, with a Cauchy distribution (F_m, 0.1), where they use a fixed scale parameter 0.1 and an adaptive location parameter F_m. The Gaussian (C_{rm}, 0.1) has a fixed standard deviation of 0.1 and a mean of C_{rm}.

Liu et al. [21] investigated operator adaptation in EP both at the population and individual levels. The operator adaptation at the population level aims to update the rates of the operator based on the operator performance over the entire population during the evolution [21].

In [17], a mixed mutation strategy with a local fitness landscape was proposed: In these strategies, a local fitness landscape was used as a key factor to determine the mutational behaviour. In [16], ensemble strategies with adaptive EP was proposed. In this work, EP with an ensemble of Gaussian and Cauchy mutation operators was proposed where each mutation operator has its own population and parameters. In [12], mutation operators with different parameters were selected in each EP generation according to the step size of the jumps by individuals. In this strategy, the size of the jumps the mutation operator can make keeps changing during the EP evolutionary process. These studies can be considered as human designed adaptive mutation strategies. Liu [21] proposed that operator adaptation in EP can be investigated at both the population and individual levels. In [12], we introduced a mutation strategy for EP, generating long step size variants at the beginning and short step size variants later on in the search.

Regardless of the type of adaptive or non-adaptive mutation operators used in state-of-the-art algorithms, the essence of the adaptive mutation operator is the following:

- Use different mutation operators, or a combination of them, for each generation.
- Change the jump sizes for different generations according to the feedback from the current generation or historical information (i.e. each EP generation has a best fitness value, the best fitness values of each generation can be stored in an array, the array fitness values are an example of historical information).

The proposed hyper-heuristic GP algorithm

In this section, we describe how GP is used to build EP mutation operators. In previous work, the GP framework successfully designed non-adaptive mutation operators for EP [7,9]. In [7,9], Hong et al. used GP as an offline learning hyper-heuristic to automatically design a mutation operator for EP and automatically design more general mutation operators for EP. In [7], a group of mutation operators were designed for function classes. In [9], function classes were classified into several groups, and each group is assigned

an independent automatically designed mutation operator. In both works, the automatically designed mutation operator is fixed at each EP generation. The automatically designed mutation operator is a probability distribution. Once the probability distribution is automatically designed, the form of the equation with the values are fixed when an EP is used; thus, the search region is fixed at each EP generation.

However, it is obvious that dynamically updating the probability distribution during the evolutionary process can lead to a more efficient search. In this study, we employ the improved framework in Fig. 1 by more creative GP terminal settings, while using GP as an *offline* hyper-heuristic to automatically design *adaptive* mutation operators for EP on specific function classes. In the framework, GP sits at the hyper-level to generate a piece of code (which acts as a mutation operator by generating random numbers according to a probability distribution), and secondly, generate functions from the function classes to optimise. The automatically generated program is actually an automatically designed probability distribution with adaptive factors. For example, CUR_MIN_X represents the absolute minimum value of the best individual in an EP run up to the current generation, which is updated at each EP generation; thus, the jump sizes of the ADAM change dynamically at different EP generations. The settings in Table 1 are adaptive factors that exist in the mutation operator. At the base-level, EP optimises functions and EP is treated as a fitness function by GP in the framework. The fitness value used in GP is the value calculated, averaged over nine runs by EP.

In Fig. 1, the blue arrow between 'Function class' and 'Function to optimise' represents functions are generated from the function class. The blue arrow between 'Function to optimise' and 'Evolutionary programming' represents functions that are taken by EP and optimised. The blue arrow between 'Evolutionary programming' and 'Adaptive mutation operator generator' represents the insertion of an ADAM which is inserted into EP. The adaptive mutation operator is generated by GP and inserted into EP where it is tested on a set of functions.

In the experiments, we perform comparisons of the two types of automatically designed mutation operators: A non-adaptive mutation operator (ADM), which is a random number generator according to a fixed probability distribution, and an adaptive mutation operator (ADAM), which is a random number generator according to a probability distribution that dynamically changes during the EP run. This means that the mutation operator could be different at each EP generation. To identify the hypothesis proposed in this study, in contrast with the framework we proposed in [7,9], we propose two significant improvements:

– The adaptive factors, are proposed and recalculated at each EP generation, added to the terminal set for GP.

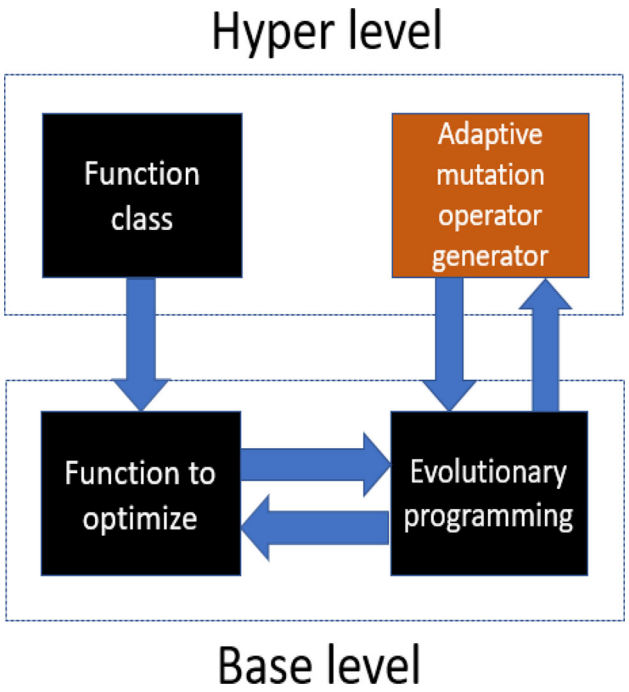

Fig. 1 Overview of the improved hyper-heuristic framework proposed, GP sits at the hyper-level to generate heuristics, EP sits at the base-level to validate the heuristics

– In [7–9], the GP framework uses EP as a fitness function, which is used to evaluate the performance of GP individuals, through non-adaptive mutation operators. In this work, the adaptive factor values are used as part of the adaptive mutation operator for EP. Due to the adaptive factors added, EP needs to calculate and update values of adaptive factors at each generation. The updated values of adaptive factors lead to the probability distribution of the possible step sizes to which a mutation operator can make changes.

Experimental design

We call a random number generator produced by GP without adaptive factors *an automatically designed mutation operator* (ADM) [7], and a random number generator produced by GP with adaptive factors *an automatically designed mutation operator* (ADAM). The experiment is designed to test the following hypothesis: ADAM can search different regions at different EP generations. The ADAM achieves better performance than an ADM for EP on specific function classes. Thus, the adaptive factors are calculated and used by EP when evolution is being processed. In the experiment, 168 functions are generated from each function class. Eighteen functions are used in the training stage, 50 functions are used in the testing stage, 100 functions are used in the exchange testing (50 for ADAM and 50 for ADM).

Table 1 The adaptive factors included in the GP terminal set

Symbol	Terminal
$N(\mu, \sigma^2)$	Normal distribution
U	$\sim$[0, 3]
GEN	Current EP generation number
CUR_MIN_X	Absolute minimum value of the best individual in an EP run
CUR_MAX_X	Absolute maximum value of the best individual in an EP run
CUR_STD_X	Standard deviation of the best individual in an EP run
CUR_MIN_Y	Best fitness value of all individuals in an EP run
CUR_MAX_Y	Worst fitness value of all individuals in an EP run
CUR_STD_Y	Standard deviation of the fitness values in an EP run
$N(\mu, CUR\_MIN\_X)$	CUR_MIN_X used as σ^2 for normal distribution
$N(\mu, CUR\_MAX\_X)$	CUR_MAX_X used as σ^2 for the normal distribution
$N(\mu, CUR\_STD\_X)$	CUR_STD_X used as σ^2 for the normal distribution
$N(\mu, CUR\_MIN\_Y)$	CUR_MIN_Y used as σ^2 for the normal distribution
$N(\mu, CUR\_MAX\_Y)$	CUR_MAX_Y used as σ^2 for the normal distribution
$N(\mu, CUR\_STD\_Y)$	CUR_STD_Y used as σ^2 for the normal distribution

The training stage

The algorithm proposed in [7,9] demonstrates automatically designed non-adaptive mutation operators have better performance than human designed mutation operators. In these experiments, we train mutation operators for each function class with two different terminal sets. In one experiment, the GP terminal set contains adaptive factors (see Table 1) that are generated by EP. In the other experiment, the terminal set of GP excludes adaptive factors (see Table 4). Both training processes use the same set of functions generated from function classes. In the experiments, ADAM denotes adaptive factors with automatically designed mutation operators. The tailored ADM automatically designed for the function class F_g is represented with ADM_g, where g is the function class index. ADM_g is called a dedicated ADM for the function class F_g. The tailored ADAM automatically designed for a function class F_g is represented by ADAM_g, where g is the function class index. ADAM_g is called a dedicated ADAM for function class F_g.

In the experiments, where a subtree crossover is applied, nodes are randomly selected from both parent trees, and the related branches are exchanged creating two offspring. A one-point mutation with the grow initialisation method is applied in GP to generate a new offspring [1,38]. During training, the GP fitness value is derived from the averaged fitness values of the nine EP runs. Each ADM_g or ADAM_g is used as an EP mutation operator on nine functions drawn from a given function class. The fitness value of an ADM_g or ADAM_g is the average of the best values obtained in each of the individual nine EP runs on the nine functions. We use the same nine functions from each function class for the entire GP run on a given function class. In general, for one function class, 18 functions are taken for training, nine of which are used to calculate the fitness value, and nine others to monitor over-fitting.

Table 2 Parameter settings for EP

Parameter	Settings
Population size	100
Tournament size	10
The initial value of the strategy parameter	3.0

To ensure that the experimental data is more traceable and easier to compare, the function classes and number of generations for each function class used in this study follow the settings in study [9]. The EP parameters follow [11,14,23], and are presented in Table 2: the population size is set to 100, the tournament size is set to 10, the initial standard deviation is set to 3.0. The settings for the dimension n, and the number of domains S are listed in Table 6. To reduce the cost of training, in our experiment the maximum number of EP generations was set to 1000 for F_1–F_{13} and F_{15}. The maximum number of EP generations was set to 100 for F_{14} and F_{16}–F_{23}.

The terminal set includes the adaptive factors outlined in Table 1, and excluding the adaptive factors outlined in Table 4. In both tables, N is a normal distribution for which the value of μ is a random number in $[-2, 2]$. This value may cause the designed mutation operator to not be Y-axis symmetric. σ^2 is a random number in $[0, 5]$. U is the uniform distribution in the range $[0, 3]$. In Table 1, GEN is the current EP generation index: It is an integer in the range [1, 1000] or [1, 100], which depends on the function classes we selected. CUR_MIN_Y is the best fitness value

Table 3 Parameter settings for GP

Parameter	Value
Population size	20
Initial number of generations	25
Crossover proportion	45%
Mutation proportion	45%
Reproduction proportion	10%
Selection method	Lexictour [36]
Depthnodes	2 [37]
Maximum initial size of tree	28
Maximum size of tree	512
Number of EP iterations	9

Table 4 Terminal set of GP without the adaptive factors

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

Table 5 Function set for GP

Symbol	Function	Arity
+	Addition	2
−	Subtraction	2
×	Multiplication	2
÷	Protected division	2
power	Power	2
exp	Exponential function	1
abs	Absolute	1

of the individuals up to the current generation, and the framework records the following for a given EP individual (each individual is taken as a pair of real-valued vectors in EP) [23]: CUR_MIN_X, as the absolute minimum value for this individual, CUR_MAX_X, as the absolute maximum value for this individual, and CUR_STD_X as the standard deviation of this individual. CUR_MAX_Y is the worst fitness value of the individuals in the current generation, and CUR_STD_Y is the standard deviation of all fitness values in the current generation. All these values constitute useful information, and will change during the EP runs.

The GP parameter settings are listed in Table 3. $depthnodes$ is set to 2, indicating that restrictions are to be applied to the tree size (number of nodes) [36]. The GP function sets are listed in Table 5. The other GP settings are: The population size is 20, and the maximum number of generations is 25. The algorithm framework to automatically design ADAM is described in Algorithms 1 and 2.

Testing the ADAM$_g$, ADM$_g$, and human designed mutation operators

We use ADAMs (in Table 9), ADMs (in Table 10), and human designed EP mutation operators, and test them on each function class F_g. For each ADAM or ADM, we record 50 values from 50 independent EP runs, each being the lowest value over all EP generations, and we then average them: This is called the mean best value. The results of testing in Tables 7, 8, and 11 are based on the same 50 functions generated from each function class F_g. Thus, in total 68 functions are used for both training and testing stages.

Algorithm 1 Algorithm Framework to Generate ADAM for A Given Function Class.

Initial parameters of GP including MaxGPGen, function set and terminal set;
Initial the GPPOP;
Set p = 0; Set m = 0; Set N = 9;
Generate N function instances from a function class;
while p < GPpopsize **do**
  while m < N **do**
    Evaluate fitness value bestEPFit$_m$ by EP for GPPOP$_p$ for mth function instance;
    m = m + 1;
  end while
  Calculate mean value GPFit$_p$ = $\sum_{m=0}^{N-1} bestEPFit_m/n$ as fitness value for GPPOP$_p$;
  p = p + 1;
end while
Record best GP fitness value $bestGPFit$ and best GP individual $bestGPInd$;
Set g = 0;
while g ≤ MaxGPGen **do**
  g = g + 1;
  Set p = 0; Set m = 0;
  while p < GPpopsize **do**
    while m < N **do**
      Evaluate fitness value bestEPFit$_m$ by EP for GPPOP$_p$ for mth function instance;
      m = m + 1;
    end while
    Calculate mean value GPFit$_p$ = $\sum_{m=0}^{n-1} bestEPFit_m/n$ as fitness value for GPPOP$_p$;
    if GPFit$_p$ < $bestGPFit$ **then**
      Update best GP individual $bestGPInd$;
      Update best GP fitness value $bestGPFit$;
    end if
    p = p + 1;
  end while
  Crossover, mutation or reproduction for next GPPOP;
end while
return $bestGPInd$;

The testing stage

In [11,13,17,23], a suite of 23 benchmark functions are commonly used to test the different EP variants, where 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 [11].

In this study, we built function classes based on the following function classes [7]. We use function classes instead of single functions for benchmark function optimisation. In other words, we are specially designing an ADAM for a class of functions.

The set of functions can be considered as a set of training instances of offline learning hyper-heuristics. The framework generates functions (instances) from the function class to train an ADAM or ADM. We then draw an independent set of functions (unseen instances) from the function class to test the ADAMs and ADMs produced from the training stage.

Based on the 23 functions, we have constructed 23 function classes in Table 6, with the index of each function class corresponding to the original functions of [23]. In Table 6, a_i, b_i, and c_i are uniformly distributed in range [1, 2], [−1, 1], and [−1, 1], respectively. The definition of the function class has been proposed in [7,9]. In this study, we use the symbol f_g for a function, and F_g for a function class.

To observe the performance of the ADAM$_g$, ADM$_g$ and human designed mutation operators, we tested the functions drawn from on F_g. The mean best values and standard deviations are listed in Table 7. The highest mean best values are in boldface. Each mean value is averaged over 50 runs. To improve the observation, we retain more decimal places for F_6, F_{16}, F_{17}, F_{18}, and F_{19}.

We also performed the Wilcoxon signed-rank test for ADAM$_g$ versus ADM$_g$, Lévy (1.0, 1.2, 1.4, 1.6, 1.8, 2.0), the results of which are given in Table 8. The Lévy distribution, where $\alpha = 1.0$ is a Cauchy distribution, whereas $\alpha = 2.0$ yields a Gaussian distribution [14]. Due to diverse features of different distributions, they have different performance on different benchmark functions.

Algorithm 2 EP Evaluation with Adaptive Factors for GPPOP$_p$ on mth Function Instance.

1: Initial parameters of EP including MaxEPGen, EPpopsize and dimension size D;
2: Generate the initial population of μ individuals. Each individual is taken as a pair of real-valued vectors, (x_i, η_i), $\forall i \in \{1, \ldots, \mu\}$;
3: Evaluate the fitness value for each (x_i, η_i), according to mth function instance generated in Algorithm 1;
4: Calculate CUR_MIN_X, CUR_MAX_X, CUR_STD_X, CUR_MIN_Y, CUR_MAX_Y, CUR_STD_Y;
5: Set $GEN = 0$;
6: **while** $GEN \leq$ MaxEPGen **do**
7: $GEN = GEN + 1$;
8: Each parent (x_i, η_i), $\forall i \in \{1, \ldots, \mu\}$, creates a single offspring (x_i', η_i') by: for $j = 1, \ldots, D$, the factors γ and γ' have set to $(\sqrt{2\sqrt{n}})^{-1}$ and $(\sqrt{2n})^{-1}$;

$$x_i'(j) = x_i(j) + \eta_i(j) GPPOP_{p,j}$$
$$\eta'(j) = \eta_i(j) exp(\gamma' N(0,1) + \gamma N_j(0,1))$$

9: Calculate the fitness value of each offspring (x_i', η_i') by the mth function instance;
10: Conduct pairwise comparison and select the μ individuals out of the union of parents (x_i, η_i) and offspring (x_i', η_i') [23];
11: Update CUR_MIN_X, CUR_MAX_X, CUR_STD_X, CUR_MIN_Y, CUR_MAX_Y, CUR_STD_Y;
12: Update best fitness value $bestEPFit$;
13: **end while**
14: **return** $bestEPFit$;

Exchange testing for ADAM$_g$ and ADM$_g$ on each function class

The exchange testing evaluates the performance of ADAM$_g$ and ADM$_g$ on all function classes. An ADAM$_g$ designed for the function class F_g is called a tailored adaptive mutation operator, while an ADAM$_g$ tested on F_j is called a non-tailored adaptive mutation operator; here $g \neq j$. An ADM$_g$ designed for the function class F_g is called a tailored non-adaptive mutation operator, while an ADM$_g$ tested on F_j is called a non-tailored non-adaptive mutation operator. For example, ADAM$_1$ is a tailored adaptive mutation operator for F_1, but it is a non-tailored adaptive mutation operator for the function class F_2. ADM$_1$ is a tailored non-adaptive mutation operator for F_1, but it is a non-tailored non-adaptive mutation operator for the function class F_2. To observe the performance of both tailored (or dedicated) and non-tailored (or non-dedicated) ADAM$_g$ and ADM$_g$ on F_j, we tested ADAM$_g$ and ADM$_g$ on F_j over 50 runs, and a function is generated from the function class in each run. Fifty functions are used for ADAM$_g$, 50 functions are used for ADM$_g$. Thus, 100 functions are used in total. The mean best values and standard deviations are given in Tables 11, and 12. We display more decimal places for F_6, F_{16}, F_{17}, F_{18}, and F_{19}, as the results are otherwise too close to distinguish. The values of the tailored ADAM and ADM are in boldface, and the values that are better than those of the tailored ADAM and ADM are also in boldface.

Analysis of the performance of ADAM$_g$ and ADM$_g$

In this section, we compare ADAM$_g$, ADM$_g$, and the human designed mutation operators. From the experiment, both ADAM$_g$ and ADM$_g$ achieve better performance than the human designed mutation operators on F_g, and ADAM$_g$ achieves an outstanding performance on most of F_g.

In Table 7, ADAM$_7$, ADAM$_{12}$, ADAM$_{19}$, and ADAM$_{20}$ are the exceptions: the experimental results for ADM$_7$, ADM$_{12}$, and ADM$_{20}$ show that they achieve better performance than ADAM$_7$, ADAM$_{12}$, and ADAM$_{20}$; The performance of ADAM$_{19}$, and ADM$_{19}$ is the same; In this

Table 6 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	$[-100, 100]^n$
$F_4(x) = \max_i\{\mid a_i x_i \mid, 1 \le i \le n\}$	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	$[-600, 600]^n$
$F_{12}(x) = \frac{\pi}{n}\{10sin^2(\pi y_i) + a_i \sum_{i=1}^{n-1}(y_i - 1)^2[1 + 10sin^2(\pi y_{i+1}) + (y_n - 1)^2]\} + \sum_{i=1}^{n} u(x_i, 10, 100, 4)$, $y_i = 1 + \frac{1}{4}(x_i + 1)$, $u(x_i, w, k, m) = \begin{cases} k(x_i - w)^m, & x_i > w, \\ 0, & -w \le x_i \le w, \\ k(-x_i - w)^m, & x_i < -w. \end{cases}$	30	$[-50, 50]^n$
$F_{13}(x) = 0.1\{sin^2(3\pi x_1) + a_i \sum_{i=1}^{n-1}(x_i - 1)^2[1 + sin^2(3\pi x_{i+1})] + (x_n - 1)[1 + sin^2(2\pi x_n)]\} + \sum_{i=1}^{n} u(x_i, 5, 100, 4)$	30	$[-50, 50]^n$
$F_{14}(x) = [\frac{1}{500} + a_i \sum_{i=1}^{25} \frac{1}{j + \sum_{i=1}^{2}(x_i - w_{ij})^6}]^{-1}$	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	$[-5, 5]^n$
$F_{17}(x) = a_1(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6)^2 + 10b_1(1 - \frac{1}{8\pi})cosx_1 + 10$	2	$[-5, 10] \times [0, 15]$
$F_{18}(x) = a_1[1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1 x_2 + 3x_2^2)] \times [30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1 x_2 + 27x_2^2)] + b_1$	2	$[-2, 2]^n$
$F_{19}(x) = -\sum_{i=1}^{4} y_i exp[-\sum_{j=1}^{4} a_j w_{ij}(x_j - p_{ij})^2 + b_i]$	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	$[0, 10]^n$

particular case, the adaptive factors selected may not well match with F_7, F_{12}, F_{19}, and F_{20}. In the future we will analyse and collect more adaptive factors for EP, and import the new adaptive factors to design ADAMs for F_7, F_{12}, F_{19}, and F_{20}.

Table 8 lists the results of the Wilcoxon Signed-Rank Test at the 5% significance level, comparing a tailored ADAM$_g$ with an ADM$_g$ and human designed mutation operators, using Lévy distributions (with α=1.0, 1.2, 1.4, 1.6, 1.8, 2.0). In this table '$\geq$' indicates that the ADAM$_g$ performs better on F_g than ADM$_g$ or the human designed mutation operators on average. '$>$' means that the difference is statistically significant, and '$=$' means that there is no difference. In the majority of the cases, the ADAM$_g$ outperform ADM$_g$ and human designed mutation operators. '$\leq$' indicates that ADAM$_g$ performs worse on F_g than ADM$_g$.

Table 11 lists the experimental results from using all ADAMs given in Table 9, testing all function classes. 'N/A' means ADAM$_g$ may lead to generating value out of range of the machine's representation. For non-tailored function classes in certain generations. From this table we find that tailored ADAMs achieve much better performance than non-tailored ADAMs on F_j in most cases. The reason is the following: ADAMs can make different jump sizes in different EP generations; however, non-tailored ADAMs usually cannot fit in function classes, once an (non-tailored) ADAM fails to make an EP search in the entire space in the early generations, it will fall into local optima with no ability to

Table 7 Results of ADAM_g, ADM_g, and Lévy(1.0, 1.2, 1.4, 1.6, 1.8, 2.0) on the trained function classes F_g. Mean indicates the mean of the best values found over all generations over 50 runs. Std Dev denotes standard deviations

F_g		ADAM_g	ADM_g	$\alpha = 1.0$	$\alpha = 1.2$	$\alpha = 1.4$	$\alpha = 1.6$	$\alpha = 1.8$	$\alpha = 2.0$
F_1	Mean	0.36195	0.36216	0.36256	0.36786	0.42730	0.40831	0.66377	0.90638
	Std Dev	(3.19)	(3.19)	(3.19)	(3.20)	(3.17)	(3.19)	(3.28)	(3.41)
F_2	Mean	0.0186	0.0199	0.1624	0.1094	0.0887	0.0734	0.0674	0.0490
	Std Dev	(5.49E−03)	(4.29E−03)	(1.40E−02)	(9.35E−03)	(7.65E−03)	(6.11E−03)	(5.83E−03)	(6.42E−03)
F_3	Mean	626.9	2276.4	2930.0	2894.9	3409.0	3319.7	4283.9	4203.1
	Std Dev	(360.8)	(1223.6)	(1130.0)	(1493.1)	(1824.6)	(1698.1)	(2264.2)	(1896.7)
F_4	Mean	19.23	31.08	32.21	35.47	35.09	38.89	41.89	42.74
	Std Dev	(8.04)	(7.40)	(8.04)	(9.28)	(9.49)	(8.21)	(6.94)	(7.63)
F_5	Mean	− 8.14	− 7.96	− 6.97	− 6.42	− 6.80	− 6.04	− 4.96	− 3.38
	Std Dev	(6.67)	(4.70)	(5.87)	(5.90)	(6.71)	(5.42)	(6.34)	(9.75)
F_6	Mean	**0.4598161509**	0.459816151	0.459816151	0.58	1.54	38.68	199.48	378.38
	Std Dev	(3.41)	(3.41)	(3.41)	(3.39)	(4.09)	(161.12)	(718.22)	(851.03)
F_7	Mean	0.0340	**0.0306**	0.0359	0.0400	0.0502	0.0610	0.0795	0.1053
	Std Dev	(9.12E−03)	(8.20E−03)	(1.29E−02)	(1.30E−02)	(2.09E−02)	(2.65E−02)	(3.99E−02)	(5.50E−02)
F_8	Mean	− 12489.5	− 12413.0	− 10989.9	− 10528.7	− 9975.3	− 9499.0	− 9047.9	− 8064.9
	Std Dev	(137.2)	(137.8)	(404.4)	(394.9)	(494.6)	(509.8)	(709.0)	(750.1)
F_9	Mean	− 10.227058	− 10.227048	− 10.219982	− 10.22	− 10.19	− 9.81	− 8.93	− 7.74
	Std Dev	(3.16)	(3.16)	(3.16)	(3.16)	(3.12)	(3.37)	(3.53)	(3.82)
F_{10}	Mean	− 20.03	− 19.86	− 2.86	− 16.01	− 18.13	− 17.85	− 14.83	− 10.53
	Std Dev	(0.31)	(1.28)	(6.94)	(7.90)	(4.68)	(2.21)	(3.29)	(3.49)
F_{11}	Mean	**0.000015713**	0.000109	0.000963	0.008813	0.038118	0.113443	0.078892	0.107281
	Std Dev	(9.47E−06)	(6.29E−05)	(1.55E−03)	(1.74E−02)	(5.91E−02)	(2.03E−01)	(1.47E−01)	(1.83E−01)
F_{12}	Mean	0.0016	**0.0009**	0.0176	0.1154	0.5585	1.4903	2.6657	4.1759
	Std Dev	(0.008)	(0.001)	(0.067)	(0.366)	(0.585)	(1.347)	(2.290)	(3.461)
F_{13}	Mean	**0.0013**	0.0022	0.0129	0.5918	4.26	12.07	17.94	31.14
	Std Dev	(0.003)	(0.002)	(0.029)	(2.16)	(9.11)	(18.04)	(18.41)	(33.57)

Table 7 continued

F_g		ADAM$_g$	ADM$_g$	$\alpha = 1.0$	$\alpha = 1.2$	$\alpha = 1.4$	$\alpha = 1.6$	$\alpha = 1.8$	$\alpha = 2.0$
F_{14}	Mean	**0.67**	0.67	0.80	0.87	1.19	1.02	1.02	1.16
	Std Dev	(1.5E−01)	(1.5E−01)	(4.1E−01)	(4.2E−01)	(8.0E−01)	(6.3E−01)	(7.2E−01)	(8.4E−01)
F_{15}	Mean	**0.0410**	0.0417	0.0434	0.0454	0.0431	0.0443	0.0433	0.0440
	Std Dev	(1.6E−02)	(1.6E−02)	(2.1E−02)	(2.1E−02)	(2.1E−02)	(2.3E−02)	(1.8E−02)	(2.0E−02)
F_{16}	Mean	− 1.803489196	− 1.803489196	− 1.803489184	− 1.803489182	− 1.803489185	− 1.803489186	− 1.803489185	− 1.803489190
	Std Dev	(0.65)	(0.65)	(0.65)	(0.65)	(0.65)	(0.65)	(0.65)	(0.65)
F_{17}	Mean	**4.942067976**	**4.942067976**	4.942067979	4.942067978	4.942067978	4.942067979	4.942067978	4.942067977
	Std Dev	(2.767594954)	(2.767594954)	(2.767594953)	(2.767594954)	(2.767594953)	(2.767594952)	(2.767594954)	(2.767594954)
F_{18}	Mean	**4.51048489**	4.51048489	4.51048550	4.51048543	4.51048544	4.51048538	4.51048558	4.51048523
	Std Dev	(1.11731035)	(1.11731035)	(1.11731038)	(1.11731044)	(1.11731042)	(1.11731047)	(1.11731052)	(1.11731046)
F_{19}	Mean	− 5.2309675772	− 5.2309675772	− 5.22572615	− 5.23096506	− 5.23096545	− 5.22572689	− 5.21698243	− 5.22572745
	Std Dev	(1.99)	(1.99)	(1.98)	(1.99)	(1.99)	(1.98)	(1.99)	(1.98)
F_{20}	Mean	− 4.96	− 4.97	− 4.70	− 4.78	− 4.79	− 4.69	− 4.76	− 4.56
	Std Dev	(2.00)	(1.99)	(2.12)	(2.08)	(2.10)	(2.02)	(2.08)	(1.97)
F_{21}	Mean	− 2.39E+12	− 1.21E+12	− 2.46E+06	− 2.40E+06	− 2.06E+06	− 1.04E+07	− 1.72E+07	− 7.06E+06
	Std Dev	(6.56E+12)	(4.76E+12)	(5.71E+06)	(4.07E+06)	(2.45E+06)	(4.21E+07)	(1.06E+08)	(1.44E+07)
F_{22}	Mean	− 5.95E+14	− 1.47E+14	− 2.10E+06	− 2.93E+06	− 5.16E+06	− 9.68E+06	− 8.37E+06	− 1.06E+07
	Std Dev	(1.85E+15)	(1.04E+15)	(3.39E+06)	(5.56E+06)	(1.42E+07)	(5.41E+07)	(1.92E+07)	(2.16E+07)
F_{23}	Mean	− 1.06E+15	− 5.99E+10	− 2.69E+06	− 4.84E+06	− 4.93E+06	− 1.40E+07	− 2.78E+07	− 1.13E+07
	Std Dev	(3.47E+15)	(1.90E+11)	(4.72E+06)	(1.08E+07)	(1.67E+07)	(8.43E+07)	(1.30E+08)	(4.09E+07)

Table 8 Wilcoxon signed-rank test of $ADAM_g$ versus ADM_g and Lévy distribution (with α = 1.0, 1.2, 1.4, 1.6, 1.8, 2.0)

F_g	ADM_g	$\alpha = 1.0$	$\alpha = 1.2$	$\alpha = 1.4$	$\alpha = 1.6$	$\alpha = 1.8$	$\alpha = 2.0$
F_1	$\geq$	$\geq$	$\geq$	$>$	$>$	$>$	$>$
F_2	$>$	$>$	$>$	$>$	$>$	$>$	$>$
F_3	$>$	$>$	$>$	$>$	$>$	$>$	$>$
F_4	$>$	$>$	$>$	$>$	$>$	$>$	$>$
F_5	$\geq$	$\geq$	$>$	$\geq$	$>$	$>$	$>$
F_6	$\geq$	$\geq$	$\geq$	$>$	$>$	$>$	$>$
F_7	$\geq$	$\geq$	$>$	$>$	$>$	$>$	$>$
F_8	$>$	$>$	$>$	$>$	$>$	$>$	$>$
F_9	$>$	$>$	$>$	$>$	$>$	$>$	$>$
F_{10}	$\geq$	$>$	$>$	$>$	$>$	$>$	$>$
F_{11}	$>$	$>$	$>$	$>$	$>$	$>$	$>$
F_{12}	$\leq$	$\geq$	$>$	$>$	$>$	$>$	$>$
F_{13}	$>$	$>$	$>$	$>$	$>$	$>$	$>$
F_{14}	$=$	$\geq$	$>$	$>$	$>$	$>$	$>$
F_{15}	$\geq$	$\geq$	$\geq$	$\geq$	$\geq$	$\geq$	$\geq$
F_{16}	$=$	$>$	$>$	$>$	$>$	$>$	$>$
F_{17}	$=$	$>$	$>$	$>$	$>$	$>$	$>$
F_{18}	$=$	$>$	$>$	$>$	$>$	$>$	$>$
F_{19}	$=$	$>$	$>$	$>$	$>$	$>$	$>$
F_{20}	$>$	$\geq$	$\geq$	$\geq$	$>$	$\geq$	$\geq$
F_{21}	$\geq$	$\geq$	$\geq$	$\geq$	$\geq$	$\geq$	$\geq$
F_{22}	$\geq$	$\geq$	$\geq$	$\geq$	$\geq$	$\geq$	$\geq$
F_{23}	$\geq$	$\geq$	$\geq$	$\geq$	$\geq$	$\geq$	$\geq$

Table 9 ADAMs discovered by GP

ADAM	Best ADAM found by GP
$ADAM_1$	÷(N(0.90415,GEN),N(0.56635,GEN))
$ADAM_2$	N(0.0058876,CUR_MIN_Y)
$ADAM_3$	×(GEN ÷(×(÷(×(GEN N(1.6806,CUR_MAX_X)) CUR_MAX_Y) N(-0.70715,CUR_MAX_X)) GEN))
$ADAM_4$	÷(N(0.11157,GEN) N(-1.3787,CUR_MAX_X))
$ADAM_5$	÷(N(-0.96806,CUR_MAX_X) −(N(-0.62258,CUR_MAX_X) N(-1.9713,1.6321)))
$ADAM_6$	−(N(-1.4874,4.6908) ÷(CUR_MIN_Y N(0.076592,CUR_STD_Y)))
$ADAM_7$	÷(CUR_MAX_Y N(0.11449,CUR_MAX_Y))
$ADAM_8$	+(exp(N(-1.1665,4.4708)) +(÷(CUR_MAX_Y plus(−(×(N(-1.3502,1.0327) ÷(exp(N(-1.8983,1.7794)) CUR_STD_Y)) −(×(N(-0.017172,3.3643) ÷(exp(N(0.41282,1.5366)) CUR_STD_Y)) CUR_STD_Y)),+(÷(GEN CUR_MAX_Y) N(-0.9974,GEN)))) N(0.6046,0.33266)))
$ADAM_9$	×(CUR_MAX_X ×(÷(N(-1.6727,GEN) N(-0.47283,GEN)) ÷(CUR_MAX_X +(CUR_MAX_X CUR_MAX_Y))))
$ADAM_{10}$	÷(-(N(-0.52805,3.0285) ×(abs(N(0.27085,2.434)) CUR_STD_Y)) ×(N(-1.2256,2.4922) abs(exp(exp(CUR_STD_Y)))))
$ADAM_{11}$	÷(÷(CUR_MAX_Y,CUR_STD_Y) N(-1.1546,4.3386))

Table 9 continued

ADAM	Best ADAM found by GP
$ADAM_{12}$	×(abs(CUR_STD_X) ×(N(-1.9085,CUR_MAX_X) ÷(N(-1.5148,CUR_MAX_X) N(-0.10725,CUR_MAX_X))))
$ADAM_{13}$	÷(÷(CUR_STD_X N(-0.17603,CUR_STD_X)) +(N(-0.66198,2.4311) 1.8755))
$ADAM_{14}$	×(N(0.16457,CUR_STD_X),power(N(-0.4513,CUR_MAX_X),N(0.75071,1.674)))
$ADAM_{15}$	×(CUR_MIN_Y power(−(N(-1.568,CUR_MAX_X) abs(CUR_MAX_X)),abs(N(-0.34848,CUR_STD_X,1,1))))
$ADAM_{16}$	÷(power(CUR_MIN_X −(0.67663 N(-1.1373,CUR_MAX_X))) ×(×(power(N(0.15272,CUR_MIN_X) N(-1.8514,CUR_MAX_X)) N(0.3968,CUR_MAX_X)) abs(exp(CUR_MIN_Y))))
$ADAM_{17}$	÷(÷(N(-1.7474,CUR_MIN_X) N(0.53761,CUR_MAX_Y)) GEN)
$ADAM_{18}$	÷(CUR_MIN_X N(1.2301,CUR_MIN_Y))
$ADAM_{19}$	÷(÷(CUR_STD_X N(0.062251,4.7214)) +(N(1.2588,4.8929) ÷(−(×(N(-0.92113,3.8014) ÷(-(0,CUR_MIN_Y) N(-0.28818,0.71497))) N(0.9943,0.61349)),CUR_STD_X)))
$ADAM_{20}$	÷(CUR_MAX_X,N(0.28085,GEN))
$ADAM_{21}$	÷(÷(÷(÷(÷(CUR_STD_X 2.5725) N(-0.95096,GEN)) N(-0.35258,GEN)) N(-1.9889,GEN)) N(-1.069,GEN))
$ADAM_{22}$	÷(−(power(power(CUR_MAX_Y exp(−exp(CUR_STD_X))) abs(CUR_STD_X)) power(power(CUR_MAX_Y N(-0.77135,0.6816)) N(-0.89184,0.5241))) power(minus(N(0.21547,1.8121) ,CUR_MIN_Y) exp(exp(N(0.58382,2.6967)))))
$ADAM_{23}$	÷(−(−(abs(2.3552) 0.70777) 1) ×(exp(power(1.4883 power(N(1.0712,CUR_STD_Y) 2.5113))) exp(abs(normrnd(0.59628,GEN)))))

subsequently jump out and escape them. As the values generated by ADAM are probably too short to allow individuals to jump out. In the future we can search for smarter adaptive factors to design ADAMs to avoid falling into local optima earlier, or establish a mechanism to avoid this problem, such as importing long step size mutation operators occasionally in a later EP generation.

The results in Tables 11, and 12 demonstrate that $ADAM_1$, $ADAM_3$, $ADAM_4$, $ADAM_5$, $ADAM_6$, $ADAM_8$, $ADAM_9$, $ADAM_{10}$, $ADAM_{11}$, $ADAM_{14}$, $ADAM_{16}$, $ADAM_{17}$, $ADAM_{18}$, $ADAM_{19}$, $ADAM_{21}$, $ADAM_{22}$, and $ADAM_{13}$ achieve the best performance when acting as tailored operators, i.e. acting on their own F_g. $ADAM_2$, $ADAM_7$, $ADAM_{12}$, $ADAM_{13}$, $ADAM_{15}$, and $ADAM_{20}$ achieve satisfactory performance, albeit not the best.

The large variation in a single mutation enables EP to globally search a wider area of the search space [14]. Researchers attempted to design mutation strategies that can generate larger variations, especially when local optima have deep and wide basins, in early EP generations, and smaller variations in the later EP generations. Otherwise, mutation strategies can generate smaller variations in early EP generations and reduced size large variations, still have chances to jump out of local optima in the later evolutionary process if they fall into early generation local optima, in the later EP generations. The ADAM can perform well on functions generated from function classes.

In Figs. 2, 3 and 4, $ADAM_3$, $ADAM_4$, $ADAM_{11}$, $ADAM_{13}$, $ADAM_{21}$, $ADAM_{22}$, and $ADAM_{23}$ achieve significant performance for EP. $ADAM_2$, $ADAM_6$, $ADAM_{12}$, and $ADAM_{14}$ do not perform well in the early EP generations. However, $ADAM_2$, $ADAM_6$, $ADAM_{12}$, and $ADAM_{14}$ lead to an acceleration in the convergence rate in the later EP generations due to the significant effectiveness of the EP adaptive factors. The experiments also demonstrate that these adaptive factors have positive impact, especially in later EP generations.

Let us further analyse the ADAMs. In Fig. 3, $ADAM_{12}$ does not perform well in the early EP generations and shows the worst performance among all mutation operators in early generations (where the generation index is less than approximately 500). This situation changes in the later generations. The adaptive factors in $ADAM_{12}$ does not help EP establish efficient convergence in the early generations. The values in Table 13 show random numbers generated by $ADAM_{12}$ that are large in early EP generations, and much smaller in the later generations. It is obvious that the fit for $ADAM_{12}$ improves in the later EP generations, as the jump sizes the mutation operator can make is significantly more precise due to the variability in mutation operators. We also notice that the processing shapes of ADM_{22} appear as 'punctuated equilibria', with F_{22} as a multimodal function class with a few local optima. This is because the evolutionary process makes no improvement in some generations.

Table 10 ADMs discovered by GP

ADM	Best ADM found by GP
ADM_1	×(÷(+(÷(+(N(-0.3842,2.5211) 1.5461) N(-1.6217,1.3702)) N(1.3287,4.6137)) N(-0.54092,4.4922)) 0.43652)
ADM_2	÷(÷(exp(N(-0.38136,0.86466)) exp(N(1.5842,0.84216))) N(0.21558,3.8436))
ADM_3	−(−(−(N(1.535,1.3136) N(0.14157,1.5186)) N(0.11891,2.4488)) −(N(1.9467,2.6814) N(0.69438,1.4041)))
ADM_4	÷(÷(− (÷(abs(N(1.0534,1.9781)) N(-0.83315,1.0421)) N(-0.76623,1.4908)) −(abs(N(0.047489,1.7402)) abs(÷(abs(N(1.8085,4.8537)) exp(N(-0.28834,1.8282)))))) abs(N(1.865,0.91793)))
ADM_5	−(N(1.5968,4.6973) −(×(×(N(1.7932,1.3455) N(0.86795,0.32284)) N(1.3364,3.623)) ×(N(-0.11477,4.0585) N(-1.6811,2.0561))))
ADM_6	power(−(abs(N(-0.21175,0.7145)) exp(1.0128)) N(0.8463,3.0375))
ADM_7	+(1 ÷(N(0.086792,1.8674) abs(0.19647)))
ADM_8	+(−(N(1.0741,4.0709) +(N(-0.85632,3.3886) exp(1.1719))) −(exp(N(-0.71554,3.1857)) abs(N(1.5423,4.2601))))
ADM_9	÷(÷(÷(÷(abs(N(-1.346,3.5533)) abs(exp(1.9001))) N(-0.99667,2.831)) abs(exp(1.5705))) N(1.6685,4.6269))
ADM_{10}	÷(÷(exp(1) N(1.085,3.0345)) ×(exp(exp(−N(-0.516,0.081374))) N(1.5994,2.5965)))
ADM_{11}	÷(1 N(-0.059745,0.11982))
ADM_{12}	÷(+(N(-0.3992,4.7626) +(+(÷(N(0.40938,1.2675) ÷(+(N(0.14379,4.7507) N(-1.7051,1.4454)) N(-1.7518,0.68855))) 2.0986) ÷(N(0.31538,0.73037) N(-1.8513,0.79594)))) +(N(-0.4157,4.2054) N(0.84778,4.1008)))
ADM_{13}	÷(−(N(-0.014005,1.326) N(-0.046516,2.7121)) ×(N(0.079839,2.6104) N(-0.92208,3.1112)))
ADM_{14}	−(N(0.24476,0.87804) power(N(-1.9097,4.9483) N(0.90134,2.2834)))
ADM_{15}	×(power(+(N(-1.7387,0.35176) N(-1.1476,2.7704)) −(×(power(N(1.2007,2.8697) ×(power(N(-1.3949,1.2935) 1.2768) 1.5861)) N(1.5794,3.4152)) N(-0.84802,4.8521))) N(0.38249,1.5269))
ADM_{16}	÷(−(abs(−(N(0.23803,2.3456) 1)) 1) times(N(-1.7843,3.2233),×(power(−(N(-1.7408,4.4291) exp(N(1.3749,0.065262))) 2.8202),−(N(0.012852,1.9296) abs(−(abs(−(abs(N(-0.73909,0.48735)) 1)) 1))))))
ADM_{17}	÷(+(N(0.83974,3.9232) N(-1.4048,4.0869)) ÷(exp(power(exp(0.2235) ÷(1.0388 N(1.3618,3.822)))) ÷(×(0.049985 N(0.84732,0.95156)) exp(exp(0.39089)))))
ADM_{18}	÷(÷(exp(N(-0.27187,4.8309)) +(N(0.10832,3.7946) ÷(exp(×(0.67132 +(N(0.59398,3.6087) exp(abs(2.2088))))) ×(N(-0.035766,2.1464) N(-0.7089,0.76699))))) ×(N(-0.68692,2.8237) +(exp(times(1.3702,+(N(1.2854,0.12994) exp(abs(1.2545))))) exp(abs(0.15397)))))
ADM_{19}	÷(×(0.021879 ÷(exp(N(-1.8043,2.246)) 1.3902)) minus(2.0914,N(1.9211,3.3417)))
ADM_{20}	÷(abs(÷(1 +(power(2.4325 2.45) 1.8624))) times(2.497,N(-0.58376,4.5423)))
ADM_{21}	÷(×(−÷(+(−(N(-0.04211,0.96206) N(0.39053,1.8724)) N(0.3999,0.94149)) +(N(-1.5001,1.9898) 0.55696)) abs(abs(abs(N(-0.62885,3.1442))))) +(power(exp(abs(−(N(-1.5485,2.9356) ÷(+(abs(+(N(-1.9603,4.9185) ×(abs(N(-1.4353,3.2339)) abs(abs(abs(N(-0.14118,0.60787))))))) +(N(-0.51722,4.3085) 0.81261)) plus(N(1.0337,4.685),0.999))))) exp(2.8301)) +(0.17287 N(0.7288,0.29907))))
ADM_{22}	exp(−exp(abs(−(0.25891 abs(exp(exp(N(1.0338,0.63885)))))))
ADM_{23}	÷(power(0.22581 0) power(power(abs(1.1746),÷(abs(N(-1.7643,4.076)) abs(N(-0.7083,0.10633)))) +(exp(plus(2.5618,N(1.9801,2.0706))) abs(exp(÷(÷(N(-1.5338,2.7202) +(power(N(1.5824, 2.5102) abs(N(1.4242,4.6976))) abs(exp(÷(N(-0.46107,4.8761) 2.4629))))) 0.079465))))

Table 11 The mean best values (averaged over 50 runs) and standard deviations (in the parentheses) for $ADAM_g$ tested on different function classes, the fitness value of the $ADAM_g$ is in bold, and those other fitness values that are better than the fitness value of the $ADAM_g$ are also in bold

F_g	$ADAM_1$	$ADAM_2$	$ADAM_3$	$ADAM_4$	$ADAM_5$	$ADAM_6$	$ADAM_7$	$ADAM_8$	$ADAM_9$	$ADAM_{10}$	$ADAM_{11}$	$ADAM_{12}$
F_1	**0.36195**	89516.0	7.25	33.92	0.36473	0.58641	1.27	87869.1	0.45214	2999.7	0.40740	0.40650
	(3.19)	(10882.3)	(3.35)	(7.56)	(3.19)	(3.21)	(2.08)	(9440.0)	(3.22)	(4143.9)	(3.21)	(3.18)
F_2	0.15954	**0.0186**	134.4	26.80	0.18248	1.98	0.20241	391.7	2.33	73.87	0.69899	0.55778
	(0.02)	(0.005)	(22.38)	(3.69)	(0.02)	(0.36)	(0.02)	(1565.7)	(16.35)	(38.68)	(0.09)	(0.06)
F_3	3197.3	153771.2	**626.9**	1618.0	3018.9	4955.7	2955.1	150412.5	4824.8	21917.5	1572.0	153955.8
	(1413.2)	(37461.4)	(360.8)	(852.3)	(1635.7)	(5118.5)	(1682.1)	(40364.8)	(2381.3)	(9665.3)	(786.5)	(40738.9)
F_4	32.53	88.99	92.56	**19.23**	31.28	92.70	31.24	91.10	88.06	46.78	83.93	92.06
	(8.26)	(11.28)	(4.07)	(8.04)	(9.06)	(5.48)	(8.87)	(3.92)	(13.19)	(10.49)	(17.60)	(3.89)
F_5	− 7.40	3077230.1	854.3	36.2	− 8.14	− 3.68	− 1.15	3170684.0	427.6	6252.9	− 7.16	− 3.18
	(6.26)	(616812.0)	(2144.1)	(53.04)	(6.67)	(13.96)	(2.05)	(588781.5)	(1071.1)	(27237.8)	(6.51)	(11.69)
F_6	**0.4598161509**	136386.6	13.34	54.42	**0.4598161509**	**0.4598161509**	0.979816151	129850.4	1.20	2517.9	0.739816151	234.9
	(3.41)	(17449.2)	(4.55)	(10.26)	(3.41)	(3.41)	(2.68)	(14542.1)	(3.51)	(2107.4)	(3.08)	(1655.1)
F_7	**0.0332**	0.2307	76.96	126.9	0.0872	0.1078	**0.0340**	151.1	0.0975	132.4	**0.0333**	0.0408
	(0.009)	(0.08)	(19.90)	(32.81)	(0.03)	(0.04)	(0.009)	(24.54)	(0.09)	(36.35)	(0.009)	(0.01)
F_8	− 10961.9	− 2694.3	− 4164.5	− 10937.6	− 10984.3	− 11433.7	− 3021.1	− 12489.5	− 5101.7	− 9351.5	− 11469.3	− 4131.5
	(394.3)	(481.6)	(268.3)	(384.6)	(402.3)	(310.1)	(444.3)	(137.2)	(1484.3)	(774.3)	(341.9)	(234.0)
F_9	− 10.220509	− 0.04	6.62	24.85	− 10.218177	− 9.664683	− 2.035932	239.0	− 10.227058	140.3	− 10.193753	− 10.199053
	(3.16)	(0.10)	(4.25)	(8.18)	(3.16)	(3.08)	(0.86)	(25.86)	(3.16)	(76.94)	(3.15)	(3.15)
F_{10}	− 2.07	− 0.06	− 0.07	− 3.93	− 4.85	− 0.07	− 0.07	− 0.07	− 0.07	− 20.03	− 0.07	− 0.07
	(5.96)	(0.06)	(0.31)	(6.42)	(8.48)	(0.31)	(0.31)	(0.31)	(0.31)	(0.31)	(0.31)	(0.31)
F_{11}	1.25E−03	1.06E−01	9.43E+01	1.69E−02	1.02E−03	1.36E+00	3.04E−02	2.72E+02	7.11E−05	6.08E+01	1.57E−05	8.05E+02
	(2.92E−03)	(8.21E−02)	(1.08E+02)	(4.82E−03)	(2.03E−03)	(3.60E−01)	(3.66E−03)	(5.36E+01)	(1.66E−04)	(9.08E+01)	(9.47E−06)	(9.04E+01)
F_{12}	8.03E−03	4.64E+08	1.13E+04	7.51E−01	6.60E−03	2.59E+00	3.84E−02	4.38E+08	9.72E+06	5.30E+03	1.89E−03	1.64E−03
	(2.47E−02)	(1.09E+08)	(3.57E+04)	(2.13E−01)	(2.14E−02)	(5.02E+00)	(5.50E−02)	(1.01E+08)	(2.36E+07)	(1.80E+04)	(5.07E−03)	(8.46E−03)
F_{13}	5.13E−03	4.44E+08	2.41E+04	4.72E+00	3.15E−02	1.18E+01	3.20E−02	4.43E+08	1.04E+07	2.08E+06	6.24E−03	8.45E−04
	(7.93E−03)	(1.03E+08)	(7.56E+04)	(1.09E+00)	(1.07E−01)	(3.84E+01)	(5.22E−03)	(1.05E+08)	(3.06E+07)	(1.24E+07)	(4.81E−03)	(9.62E−04)
F_{14}	0.77	1.21	0.72	0.99	0.90	0.82	0.82	2.33	0.77	1.66	0.98	0.70
	(0.38)	(0.87)	(0.25)	(0.78)	(0.60)	(0.40)	(0.36)	(1.47)	(0.34)	(2.05)	(0.70)	(0.25)

Table 11 continued

F_g	$ADAM_1$	$ADAM_2$	$ADAM_3$	$ADAM_4$	$ADAM_5$	$ADAM_6$	$ADAM_7$	$ADAM_8$	$ADAM_9$	$ADAM_{10}$	$ADAM_{11}$	$ADAM_{12}$
F_{15}	0.0418	0.0472	0.0455	0.0415	0.0437	0.0483	0.0497	0.0889	0.0412	0.0433	0.0412	0.0417
	(0.02)	(0.02)	(0.02)	(0.02)	(0.02)	(0.02)	(0.02)	(0.03)	(0.02)	(0.02)	(0.02)	(0.02)
F_{16}	− 1.803489185	− 1.044662062	− 1.803476944	− 1.803476179	− 1.803489188	− 1.803313598	− 1.78967515	− 1.47746199	− 1.80348917	− 1.803489194	− 1.803488967	− 1.803489074
	(0.65)	(0.67)	(0.65)	(0.65)	(0.65)	(0.65)	(0.65)	(0.78)	(0.65)	(0.65)	(0.65)	(0.65)
F_{17}	4.942067979	4.942167624	4.94214413	4.942068261	4.942067977	4.942286933	4.942067988	4.947497938	4.942068071	**4.942067976**	4.942068051	4.942068516
	(2.77)	(2.77)	(2.77)	(2.77)	(2.77)	(2.77)	(2.77)	(2.77)	(2.77)	(2.77)	(2.77)	(2.77)
F_{18}	4.51048546	5.67097789	4.51051369	4.55771628	4.51048519	4.51057261	4.51048796	37.32	4.51048492	4.51885579	4.51052542	4.51092337
	(1.12)	(6.17)	(1.12)	(1.15)	(1.12)	(1.12)	(1.12)	(31.98)	(1.12)	(1.13)	(1.12)	(1.12)
F_{19}	− 5.2309646210	− 4.4760907976	− 4.4959614509	− 5.2085868751	− 5.2309659241	− 5.2098022305	− 4.5629538891	− 4.8895230941	− 5.2249223288	− 5.2257281615	− 5.2309657873	− 5.2196757878
	(1.99)	(1.80)	(1.85)	(1.98)	(1.99)	(2.00)	(1.80)	(1.95)	(2.00)	(1.98)	(1.99)	(1.99)
F_{20}	− 4.61	− 2.38	− 2.28	− 2.54	− 4.57	− 3.03	− 1.95	− 2.33	− 4.88	− 4.80	− 4.81	− 4.69
	(2.08)	(1.33)	(1.19)	(1.62)	(2.12)	(1.74)	(0.85)	(1.24)	(2.11)	(2.01)	(1.91)	(1.85)
F_{21}	− 5.47E+06	− 1.35E+00	− 1.39E+00	− 2.56E+05	− 4.85E+06	− 2.52E+05	− 1.66E+00	− 1.23E+05	N/A	− 5.30E+04	− 4.42E+09	− 1.33E+06
	(1.78E+07)	(8.69E−01)	(8.42E−01)	(9.05E+05)	(2.02E+07)	(7.52E+05)	(2.49E+00)	(5.81E+05)	(N/A)	(2.16E+05)	(7.67E+09)	(3.18E+06)
F_{22}	− 2.13E+06	− 1.69E+00	− 1.88E+00	− 7.73E+04	− 3.63E+06	− 4.10E+05	− 1.48E+00	− 3.91E+04	N/A	− 1.68E+05	− 1.46E+10	− 1.38E+06
	(2.91E+06)	(8.72E−01)	(1.27E+00)	(2.16E+05)	(7.66E+06)	(2.38E+06)	(9.96E−01)	(1.68E+05)	(N/A)	(8.95E+05)	(5.63E+10)	(5.05E+06)
F_{23}	− 1.96E+06	− 2.14E+00	− 1.87E+00	− 1.39E+05	− 1.84E+06	− 8.90E+05	− 4.70E+00	− 6.18E+04	N/A	− 5.57E+04	− 1.33E+10	− 1.08E+06
	(4.51E+06)	(1.77E+00)	(9.50E−01)	(3.47E+05)	(2.78E+06)	(5.85E+06)	(1.67E+01)	(3.32E+05)	(N/A)	(1.60E+05)	(4.26E+10)	(2.37E+06)

F_g	$ADAM_{13}$	$ADAM_{14}$	$ADAM_{15}$	$ADAM_{16}$	$ADAM_{17}$	$ADAM_{18}$	$ADAM_{19}$	$ADAM_{20}$	$ADAM_{21}$	$ADAM_{22}$	$ADAM_{23}$
F_1	0.36505	88978.3	90317.9	90688.9	32948.9	26871.6	207.7	13.46	17770.9	1661.2	48813.5
	(3.19)	(10491.6)	(10000.0)	(9526.6)	(13713.4)	(13975.9)	(622.1)	(21.11)	(7570.4)	(1081.9)	(13111.4)
F_2	**0.00799**	208.6	833.9	206.5	10.94	44.30	0.19910	0.56904	51.02	9.18	92.89
	(0.005)	(286.5)	(4664.1)	(408.2)	(7.42)	(38.71)	(0.11)	(0.55)	(15.62)	(5.66)	(15.55)
F_3	3543.7	153441.6	155961.1	164136.9	51406.3	32070.1	10659.8	7354.8	34607.4	34722.7	117322.0
	(1816.1)	(27723.3)	(36030.4)	(35984.1)	(19559.2)	(14854.4)	(4202.8)	(2593.0)	(10352.5)	(7067.5)	(31686.1)
F_4	29.38	92.85	92.90	91.53	64.68	58.00	38.80	44.00	78.99	72.12	88.19
	(7.94)	(3.54)	(3.40)	(4.91)	(9.60)	(12.10)	(13.31)	(12.13)	(6.99)	(9.52)	(6.08)
F_5	− 7.41	3018121.2	3085339.4	3230995.4	954252.2	686594.2	124186.1	3.46	280662.8	55146.7	1370657.8
	(7.71)	(605972.2)	(591401.8)	(464125.6)	(610992.1)	(527747.5)	(228965.5)	(14.00)	(181873.3)	(31189.2)	(537589.1)
F_6	**0.4598161509**	139555.3	136093.7	140268.8	136263.8	59966.2	1941.4	59.06	31210.3	2497.2	71808.7
	(3.41)	(15385.4)	(18966.8)	(17543.5)	(15246.0)	(34057.2)	(3188.1)	(73.84)	(16079.8)	(1124.5)	(22933.5)
F_7	0.0443	0.1014	1.59	148.9	0.1989	2.12	0.1130	0.2837	30.41	31.27	61.38
	(0.01)	(0.04)	(0.48)	(31.13)	(0.16)	(5.74)	(0.06)	(0.19)	(12.71)	(11.48)	(22.61)

Table 11 continued

F_g	$ADAM_{13}$	$ADAM_{14}$	$ADAM_{15}$	$ADAM_{16}$	$ADAM_{17}$	$ADAM_{18}$	$ADAM_{19}$	$ADAM_{20}$	$ADAM_{21}$	$ADAM_{22}$	$ADAM_{23}$
F_8	− 11395.2	− 4871.8	− 4728.3	− 2730.8	− 2741.5	− 2695.4	− 11432.8	− 10874.1	− 9245.9	− 12179.9	− 9813.6
	(397.9)	(208.8)	(188.7)	(404.1)	(433.9)	(365.5)	(369.8)	(451.0)	(581.9)	(214.6)	(1121.1)
F_9	− 10.186914	80.82	238.3	236.5	− 1.461768	− 10.023969	5.25	− 9.862886	41.32	9.53	115.8
	(3.14)	(109.2)	(21.11)	(23.92)	(1.88)	(9.47)	(3.26)	(3.30)	(19.82)	(9.56)	(29.56)
F_{10}	− 0.07	− 0.07	− 0.07	− 0.08	− 0.52	0.05	− 0.07	− 0.07	− 0.33	− 0.07	− 1.25
	(0.31)	(0.31)	(0.31)	(0.31)	(0.15)	(0.21)	(0.31)	(0.31)	(1.08)	(0.31)	(0.65)
F_{11}	6.97E−05	8.21E+02	8.17E+02	8.18E+02	1.57E+02	5.93E+01	4.18E−04	7.97E−02	1.73E+02	4.48E+00	4.29E+02
	(1.84E−04)	(8.02E+01)	(7.77E+01)	(7.30E+01)	(9.22E+01)	(7.38E+01)	(5.43E−04)	(1.06E−01)	(8.61E+01)	(2.44E+00)	(1.12E+02)
F_{12}	8.81E−04	4.51E+08	4.70E+08	4.40E+08	1.83E+08	1.34E+08	4.01E+07	1.56E+00	1.89E+07	3.56E+06	2.45E+08
	(2.73E−03)	(9.86E+07)	(9.29E+07)	(1.15E+08)	(1.03E+08)	(9.50E+07)	(5.04E+07)	(1.40E+00)	(1.84E+07)	(3.09E+06)	(1.01E+08)
F_{13}	1.28E−03	4.61E+08	4.64E+08	4.67E+08	1.79E+08	1.42E+08	4.23E+07	1.15E+01	1.71E+07	3.95E+06	2.03E+08
	(0.003)	(9.65E+07)	(7.83E+07)	(1.11E+08)	(8.10E+07)	(7.58E+07)	(6.14E+07)	(1.37E+01)	(2.36E+07)	(3.02E+06)	(8.52E+07)
F_{14}	0.84	**0.67**	11.44	2.08	3.09	1.61	0.87	0.97	1.61	0.98	3.91
	(0.48)	(0.15)	(9.70)	(2.52)	(3.09)	(1.31)	(0.57)	(0.76)	(1.94)	(1.27)	(3.03)
F_{15}	**0.0410**	**0.0405**	**0.0410**	0.0446	0.0614	0.0887	0.0435	0.0451	0.0486	0.0427	0.0831
	(0.02)	(0.02)	(0.02)	(0.02)	(0.03)	(0.03)	(0.02)	(0.02)	(0.02)	(0.02)	(0.03)
F_{16}	− 1.803489178	− 1.803489196	− 1.803489142	− 1.803489196	− 1.788557856	− 1.044275017	− 1.803489196	− 1.803489196	− 1.803489196	− 1.803480287	− 1.786836766
	(0.65)	(0.65)	(0.65)	(0.65)	(0.65)	(0.60)	(0.65)	(0.65)	(0.65)	(0.65)	(0.67)
F_{17}	4.942067978	4.942068	4.978255727	4.975763292	**4.942067976**	4.94206814	**4.942067976**	**4.942067976**	4.942067982	**4.942067976**	4.947844412
	(2.77)	(2.77)	(2.75)	(2.78)	(2.77)	(2.77)	(2.77)	(2.77)	(2.77)	(2.77)	(2.77)
F_{18}	4.51048588	4.51048492	19.47	22.86	6.09268763	**4.51048489**	7.57949586	4.51048489	4.51048489	4.51053460	5.58333008
	(1.12)	(1.12)	(30.40)	(32.09)	(8.10)	(1.12)	(12.51)	(1.12)	(1.12)	(1.12)	(2.58)
F_{19}	− 5.2257260535	− 5.2249224419	− 5.2055306394	− 5.0052699055	− 4.5161112752	− 4.6161879909	− 5.2309675772	− 5.2309675761	− 5.2309671737	− 5.2257284204	− 5.1075787348
	(1.98)	(2.00)	(2.01)	(1.92)	(1.79)	(1.84)	(1.99)	(1.99)	(1.99)	(1.98)	(1.98)
F_{20}	− 4.70	− 4.82	− 4.18	− 3.62	− 2.39	− 2.28	− 4.95	− 4.96	− 4.91	− 4.81	− 3.66
	(1.97)	(2.10)	(1.73)	(1.39)	(1.57)	(1.22)	(1.99)	(2.00)	(1.96)	(2.12)	(1.60)
F_{21}	− 4.99E+06	− 1.11E+07	− 6.81E+02	− 6.16E+00	− 1.36E+00	− 1.52E+00	N/A	− 2.39E+07	− 2.39E+12	− 5.65E+14	N/A
	(1.88E+07)	(2.59E+07)	(1.18E+03)	(1.03E+01)	(1.04E+00)	(1.66E+00)	(N/A)	(3.47E+07)	(6.56E+12)	(2.56E+15)	(N/A)
F_{22}	− 2.95E+06	− 8.91E+06	− 5.10E+02	− 6.23E+00	− 1.67E+00	− 2.66E+00	N/A	− 1.92E+07	− 3.29E+12	− 5.95E+14	N/A
	(1.23E+07)	(2.62E+07)	(8.76E+02)	(1.03E+01)	(1.10E+00)	(3.99E+00)	(N/A)	(3.88E+07)	(8.79E+12)	(1.85E+15)	(N/A)
F_{23}	− 2.79E+06	− 3.48E+06	− 4.14E+02	− 6.17E+00	− 2.41E+00	− 2.16E+00	N/A	− 3.30E+07	− 7.53E+13	N/A	− 1.06E+15
	(7.41E+06)	(1.07E+07)	(7.19E+02)	(7.27E+00)	(1.70E+00)	(1.34E+00)	(N/A)	(5.24E+07)	(3.63E+14)	(N/A)	(3.47E+15)

Table 12 The mean best values (averaged over 50 runs) and standard deviations (in parentheses) for ADM_g tested on different function classes, the fitness value of the ADM_g is in bold, and those other fitness values that are better than the fitness value of the ADM_g are also in bold

F_g	ADM_1	ADM_2	ADM_3	ADM_4	ADM_5	ADM_6	ADM_7	ADM_8	ADM_9	ADM_{10}	ADM_{11}	ADM_{12}	ADM_{13}
F_1	**0.36216**	0.83901	0.36898	0.36777	0.37443	0.39514	0.37021	226.0	2.95700	0.37152	0.57772	0.36219	0.36457
	(3.19)	(3.19)	(3.19)	(3.19)	(3.19)	(3.19)	(3.19)	(230.2)	(7.98)	(3.18)	(3.19)	(3.19)	(3.19)
F_2	0.1156	**0.0199**	0.1813	0.2538	0.5161	2.4233	0.3764	9.9555	0.0133	0.0435	1.6605	0.1682	0.2026
	(0.0154)	(0.0043)	(0.0132)	(0.0406)	(0.0434)	(4.35)	(0.0310)	(5.38)	(0.0115)	(0.0053)	(0.1497)	(0.0189)	(0.0266)
F_3	3291.8	5520.6	**2276.4**	3429.3	2597.0	4262.2	**2126.9**	7867.6	7105.0	3891.0	4227.8	2911.7	3271.7
	(1966.9)	(2251.6)	(1223.6)	(1566.6)	(1472.7)	(1651.5)	1036.5)	(3704.1)	(2999.4)	(1934.4)	(2038.6)	(1376.9)	(1439.6)
F_4	31.93	45.98	36.97	**31.08**	52.54	33.08	42.53	82.91	50.86	39.15	78.76	33.16	**30.16**
	(7.13)	(9.65)	(8.18)	(7.40)	(12.98)	(10.75)	(11.52)	(11.32)	(8.78)	(9.54)	(14.79)	(6.62)	(6.78)
F_5	− 6.57	− 2.91	− 5.87	− 6.14	− 7.96	− 4.27	− 4.65	741.2	− 2.61	− 6.63	− 5.90	− 5.63	− 7.19
	(6.56)	(10.69)	(5.27)	(9.07)	(4.70)	(9.70)	(7.84)	(840.5)	(10.65)	(6.25)	(5.99)	(6.73)	(6.63)
F_6	0.459816151	13.58	142.4	0.459816151	0.719816151	**0.459816151**	9.12	971.9	19.08	0.539816151	0.459816151	0.459816151	0.459816151
	(3.41)	(51.13)	(562.6)	(3.41)	(3.46)	(3.41)	(46.47)	(1053.3)	(57.32)	(3.44)	(3.41)	(3.41)	(3.41)
F_7	0.0395	0.0825	0.0389	10.48	0.0246	0.0930	**0.0306**	0.1544	0.0840	0.0614	0.1335	0.0359	0.0454
	(0.01)	(0.03)	(0.01)	(33.2)	(0.008)	(0.03)	(0.008)	(0.05)	(0.03)	(0.02)	(0.03)	(0.01)	(0.02)
F_8	− 11103.2	− 10797.0	− 8443.4	− 11522.3	− 8902.7	− 11461.9	− 8834.5	− 12413.0	− 10890.8	− 11206.7	− 11055.4	− 11059.2	− 11303.4
	(349.8)	(488.8)	(575.6)	(362.0)	(587.5)	(336.4)	(526.9)	(137.8)	(469.8)	(444.8)	(350.0)	(385.5)	(353.1)
F_9	− 10.223637	− 10.202582	− 7.897187	− 6.447601	− 8.508684	97.8	− 7.995370	− 5.084338	− 10.227048	-10.226556	-9.653800	− 10.219432	− 10.210340
	(3.16)	(3.18)	(3.43)	(24.5)	(3.70)	(110.9)	(3.52)	(4.16)	(3.16)	(3.16)	(3.17)	(3.16)	(3.16)
F_{10}	− 14.04	− 19.62	− 0.46	− 1.26	− 0.07	− 0.07	− 0.07	− 0.07	− 16.75	− 19.86	− 0.07	− 2.46	− 2.46
	(9.12)	(1.05)	(2.73)	(4.67)	(0.31)	(0.31)	(0.31)	(0.31)	(4.60)	(1.28)	(0.31)	(6.42)	(6.40)
F_{11}	0.0006541	0.240736	0.010528	**0.000094**	0.000666	**0.000039**	0.003166	3.427201	0.634264	0.011474	**0.000109**	0.000400	0.000167
	(0.0013)	(0.4434)	(0.0181)	(0.0003)	(0.0011)	(0.00003)	(0.0043)	(3.15)	(1.19)	(0.0339)	(0.00006)	(0.0007)	(0.0006)
F_{12}	0.0111	0.4072	3.6690	0.0025	3.6241	0.0066	5.3063	57.62	0.5413	0.0344	0.0127	**0.0009**	**0.0005**
	(0.0405)	(0.5565)	(3.0478)	(0.0150)	(3.2298)	(0.0151)	(4.2708)	(78.35)	(0.7764)	(0.0993)	(0.0134)	(0.001)	(0.0013)
F_{13}	0.0087	1.2621	22.71	0.0044	35.83	0.0174	35.09	172.9	2.10	0.0222	0.0913	0.0068	**0.0022**
	(0.0243)	(3.0007)	(33.37)	(0.0076)	(46.79)	(0.0176)	(42.82)	(119.0)	(4.1014)	(0.0690)	(0.0880)	(0.0223)	(0.002)
F_{14}	0.85	1.05	1.14	0.70	1.29	0.68	1.26	1.27	1.49	0.91	0.84	0.90	0.81
	(0.50)	(0.88)	(0.88)	(0.22)	(1.00)	(0.19)	(0.95)	(0.85)	(1.36)	(0.55)	(0.40)	(0.57)	(0.54)
F_{15}	**0.0414**	0.0444	0.0458	**0.0399**	**0.0399**	0.0403	0.0416	0.0455	0.0463	0.0445	0.0458	0.0420	0.0434
	(0.0167)	(0.0201)	(0.0213)	(0.0169)	(0.0194)	(0.0169)	(0.0191)	(0.0198)	(0.0226)	(0.0218)	(0.0195)	(0.0195)	(0.0198)
F_{16}	− 1.803489192	− 1.803489196	− 1.80348909	− 1.803489195	− 1.803488521	− 1.80348919	− 1.803488598	− 1.803488696	− 1.803489196	− 1.803489196	− 1.803484125	− 1.803489188	− 1.803489192
	(0.65)	(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_{17}	4.942067977	**4.942067976**	4.942068007	**4.942067976**	4.942068183	4.942067977	4.942068097	4.942068082	4.942069771	**4.942067976**	4.942069099	4.942067978	4.942067977
	(2.77)	(2.77)	(2.77)	(2.77)	(2.77)	(2.77)	(2.77)	(2.77)	(2.77)	(2.77)	(2.77)	(2.77)	(2.77)
F_{18}	4.510485117	4.510484891	4.510490336	4.510484937	4.701282625	4.510485256	4.510512034	4.510518245	4.510484889	4.510484905	4.510760852	4.510485432	4.510485042
	(1.12)	(1.12)	(1.12)	(1.12)	(1.66)	(1.12)	(1.12)	(1.12)	(1.12)	(1.12)	(1.12)	(1.12)	(1.12)
F_{19}	− 5.219682565	− 5.230967562	− 5.225707061	-5.225728023	-5.224496217	− 5.230960298	− 5.210285305	− 5.223742633	− 5.230967576	− 5.230967504	− 5.130780504	− 5.230964167	− 5.230966275

Table 12 continued

F_g	ADM_1	ADM_2	ADM_3	ADM_4	ADM_5	ADM_6	ADM_7	ADM_8	ADM_9	ADM_{10}	ADM_{11}	ADM_{12}	ADM_{13}
	(1.99)	(1.99)	(1.98)	(1.98)	(1.99)	(1.99)	(2.01)	(2.00)	(1.99)	(1.99)	(2.03)	(1.99)	(1.99)
F_{20}	− 4.8226	− 4.9467	− 4.4508	− 4.7505	− 3.3490	− 4.4907	− 3.3246	− 3.5653	− 4.9805	− 4.9253	− 2.2142	− 4.6288	− 4.5997
	(2.03)	(1.99)	(1.99)	(2.02)	(1.86)	(2.12)	(2.11)	(1.81)	(1.98)	(1.99)	(1.25)	(2.08)	(2.04)
F_{21}	− 3.22E+06	− 2.55E+07	− 5.95E+05	− 8.04E+06	− 1.66E+05	− 7.46E+05	− 4.92E+05	− 1.38E+05	− 8.51E+07	− 6.23E+06	− 2.38E+05	− 2.07E+06	− 5.16E+06
	(6.37E+06)	(4.51E+07)	(2.19E+06)	(2.91E+07)	(3.11E+05)	(1.92E+06)	(1.17E+06)	(3.40E+05)	(1.65E+08)	(1.10E+07)	(1.10E+06)	(3.05E+06)	(1.41E+07)
F_{22}	− 3.27E+06	− 4.45E+07	− 8.56E+05	− 3.70E+06	− 1.62E+05	− 1.67E+06	− 3.99E+05	− 3.10E+05	− 6.56E+08	− 1.68E+07	− 7.32E+04	− 2.92E+06	− 3.03E+06
	(5.85E+06)	(1.06E+08)	(1.88E+06)	(8.64E+06)	(2.42E+05)	(4.00E+06)	(8.63E+05)	(8.26E+05)	(3.31E+09)	(5.43E+07)	(2.51E+05)	(5.57E+06)	(4.80E+06)
F_{23}	− 4.33E+06	− 9.08E+07	− 7.96E+05	− 1.42E+07	− 9.80E+05	− 1.55E+06	− 1.47E+06	− 4.11E+05	− 1.58E+08	− 2.24E+07	− 1.21E+06	− 3.52E+07	− 4.76E+06
	(1.12E+07)	(3.63E+08)	(2.54E+06)	(8.32E+07)	(3.70E+06)	(3.63E+06)	(5.60E+06)	(1.23E+06)	(4.70E+08)	(4.86E+07)	(8.27E+06)	(2.34E+08)	(1.32E+07)

F_g	ADM_{14}	ADM_{15}	ADM_{16}	ADM_{17}	ADM_{18}	ADM_{19}	ADM_{20}	ADM_{21}	ADM_{22}	ADM_{23}
F_1	11741.0	90166.9	0.41565	1227.0	5.32894	149.0	80.92	16.75	47455.0	47747.3
	(14854.7)	(9724.5)	(3.20)	(1616.0)	(18.47)	(302.8)	(179.9)	(46.57)	(14824.6)	(12339.0)
F_2	140.3	169.5	0.0967	6.8972	0.0468	0.0589	0.0419	0.0597	89.71	94.00
	(36.99)	(33.64)	(0.0536)	(6.74)	(0.0410)	(0.1149)	(0.0569)	(0.0757)	(20.25)	(13.08)
F_3	41037.2	132985.9	6440.1	14174.5	11481.7	10521.8	8676.1	10140.6	138098.2	123501.4
	(18437.4)	(18500.7)	(2126.1)	(5858.1)	(3497.2)	(3280.1)	(3650.7)	(3990.6)	(31702.5)	(35462.6)
F_4	92.06	93.11	35.67	57.30	62.65	56.89	55.31	71.57	89.45	88.13
	(4.34)	(4.04)	(6.59)	(9.96)	(11.16)	(9.49)	(10.20)	(9.92)	(4.54)	(6.01)
F_5	2914708.5	3121507.2	-3.01	65.12	-1.48	7.10	23.40	0.05	1507754.7	1365470.9
	(547819.4)	(610795.0)	(8.68)	(155.8)	(10.90)	(27.32)	(78.77)	(10.90)	(593630.6)	(497715.5)
F_6	20763.4	139104.5	0.499816151	10000.3	10.22	776.7	665.0	90.44	72827.2	70881.3
	(23678.4)	(17409.9)	(3.37)	(6873.5)	(20.17)	(1584.1)	(1308.8)	(295.8)	(17290.8)	(18120.3)
F_7	34.01	148.6	0.1042	0.4831	0.0899	0.1148	0.1299	0.0743	62.22	64.02
	(29.2)	(28.1)	(0.06)	(0.64)	(0.04)	(0.07)	(0.08)	(0.04)	(26.4)	(20.3)
F_8	− 11527.4	− 6323.7	− 12289.8	− 7943.7	− 11516.6	− 10805.7	− 10487.6	− 11597.7	− 9438.2	− 8257.3
	(512.7)	(218.5)	(197.4)	(539.1)	(356.8)	(409.0)	(470.3)	(344.7)	(1007.1)	(737.8)
F_9	222.2	237.3	− 10.222020	− 7.971963	− 10.205891	− 10.134269	− 10.003452	− 10.226557	124.5	102.3
	(60.0)	(29.2)	(3.16)	(3.99)	(3.15)	(3.20)	(3.29)	(3.16)	(36.7)	(26.4)

Table 12 continued

F_g	ADM_{14}	ADM_{15}	ADM_{16}	ADM_{17}	ADM_{18}	ADM_{19}	ADM_{20}	ADM_{21}	ADM_{22}	ADM_{23}
F_{10}	-0.07	-0.07	-7.75	-0.62	-18.37	-8.99	-6.32	-17.77	-1.07	-1.01
	(0.31)	(0.31)	(8.69)	(0.34)	(1.97)	(5.60)	(4.26)	(3.40)	(0.64)	(0.51)
F_{11}	0.081026	825.4	0.000558	29.76	3.222304	11.64	14.74	2.58	435.3	455.4
	(0.1502)	(76.28)	(0.0007)	(24.39)	(8.44)	(17.98)	(19.14)	(4.25)	(122.0)	(110.8)
F_{12}	3.86E+08	4.59E+08	0.0021	9.8042	0.2840	2.4785	3.0290	0.4335	2.20E+08	1.82E+08
	(1.05E+08)	(8.95E+07)	(0.0049)	(4.9402)	(0.3851)	(2.5108)	(2.1697)	(0.7772)	(1.03E+08)	(7.93E+07)
F_{13}	3.95E+08	4.58E+08	0.0223	94.92	1.6067	20.17	25.14	1.9100	2.27E+08	2.27E+08
	(1.25E+08)	(9.39E+07)	(0.0371)	(42.21)	(2.1779)	(17.54)	(30.60)	(4.3937)	(1.18E+08)	(9.90E+07)
F_{14}	**0.67**	**0.67**	0.77	3.70	1.44	2.27	2.63	1.52	9.33	5.03
	(0.15)	(0.15)	(0.49)	(3.33)	(1.70)	(2.64)	(2.34)	(1.80)	(14.1)	(3.35)
F_{15}	**0.0396**	**0.0417**	0.0420	0.0480	0.0507	0.0511	0.0451	0.0546	0.1200	0.0894
	(0.0171)	(0.0158)	(0.0203)	(0.0212)	(0.0264)	(0.0231)	(0.0201)	(0.0237)	(0.0728)	(0.0290)
F_{16}	− 1.803489162	− 1.803487712	− 1.803489196	− 1.803489196	− 1.803489196	− 1.803489196	− 1.803489196	− 1.803489195	− 1.395440295	− 1.766256103
	(0.65)	(0.65)	(0.65)	(0.65)	(0.65)	(0.65)	(0.65)	(0.65)	(0.70)	(0.65)
F_{17}	4.942067983	4.942068962	**4.942067976**	**4.942067976**	4.942221758	**4.942067976**	4.942067976	4.943286233	5.189129279	4.997193928
	(2.77)	(2.77)	(2.77)	(2.77)	(2.77)	(2.77)	(2.77)	(2.77)	(2.60)	(2.77)
F_{18}	4.510487416	4.510649029	4.510484889	4.510484889	**4.510484889**	4.510484889	4.510484889	7.011921768	25.05429407	9.572615511
	(1.12)	(1.12)	(1.12)	(1.12)	(1.12)	(1.12)	(1.12)	(10.1)	(27.8)	(9.83)
F_{19}	− 5.230940209	− 5.224727508	− 5.230967576	− 5.2309675769	− 5.199531198	− 5.2309675772	− 5.2309675764	− 5.211101191	− 4.894786724	− 5.025139375
	(1.99)	(1.99)	(1.99)	(1.99)	(2.02)	(1.99)	(1.99)	(2.01)	(1.96)	(2.04)
F_{20}	− 4.3757	− 4.0192	− 4.8781	− 4.9824	− 4.8152	− 4.9577	− 4.9707	− 4.9457	− 3.5593	− 3.6504
	(2.02)	(1.88)	(1.99)	(1.98)	(2.04)	(1.97)	(1.99)	(1.99)	(1.91)	(1.62)
F_{21}	− 2.31E+05	− 6.64E+02	− 2.09E+08	− 1.60E+08	− 6.08E+10	− 2.85E+08	− 8.65E+07	− 1.21E+12	− 1.78E+12	− 6.28E+11
	(5.02E+05)	(2.30E+03)	(5.47E+08)	(3.73E+08)	(3.10E+11)	(4.12E+08)	(1.17E+08)	(4.76E+12)	(1.05E+13)	(2.55E+12)
F_{22}	− 4.10E+05	− 4.19E+02	− 7.30E+08	− 2.29E+08	− 8.87E+09	− 3.79E+08	− 2.90E+08	− 1.17E+12	− 1.47E+14	− 5.37E+10
	(1.12E+06)	(1.03E+03)	(4.52E+09)	(6.61E+08)	(2.60E+10)	(6.44E+08)	(9.56E+08)	(4.45E+12)	(1.04E+15)	(1.50E+11)
F_{23}	− 1.67E+05	− 2.99E+02	− 7.26E+07	− 1.38E+08	− 1.29E+10	− 3.04E+08	− 7.63E+08	− 4.05E+12	− 7.74E+09	− 5.99E+10
	(2.71E+05)	(7.10E+02)	(1.51E+08)	(2.16E+08)	(3.42E+10)	(5.79E+08)	(3.13E+09)	(1.55E+13)	(2.63E+10)	(1.90E+11)

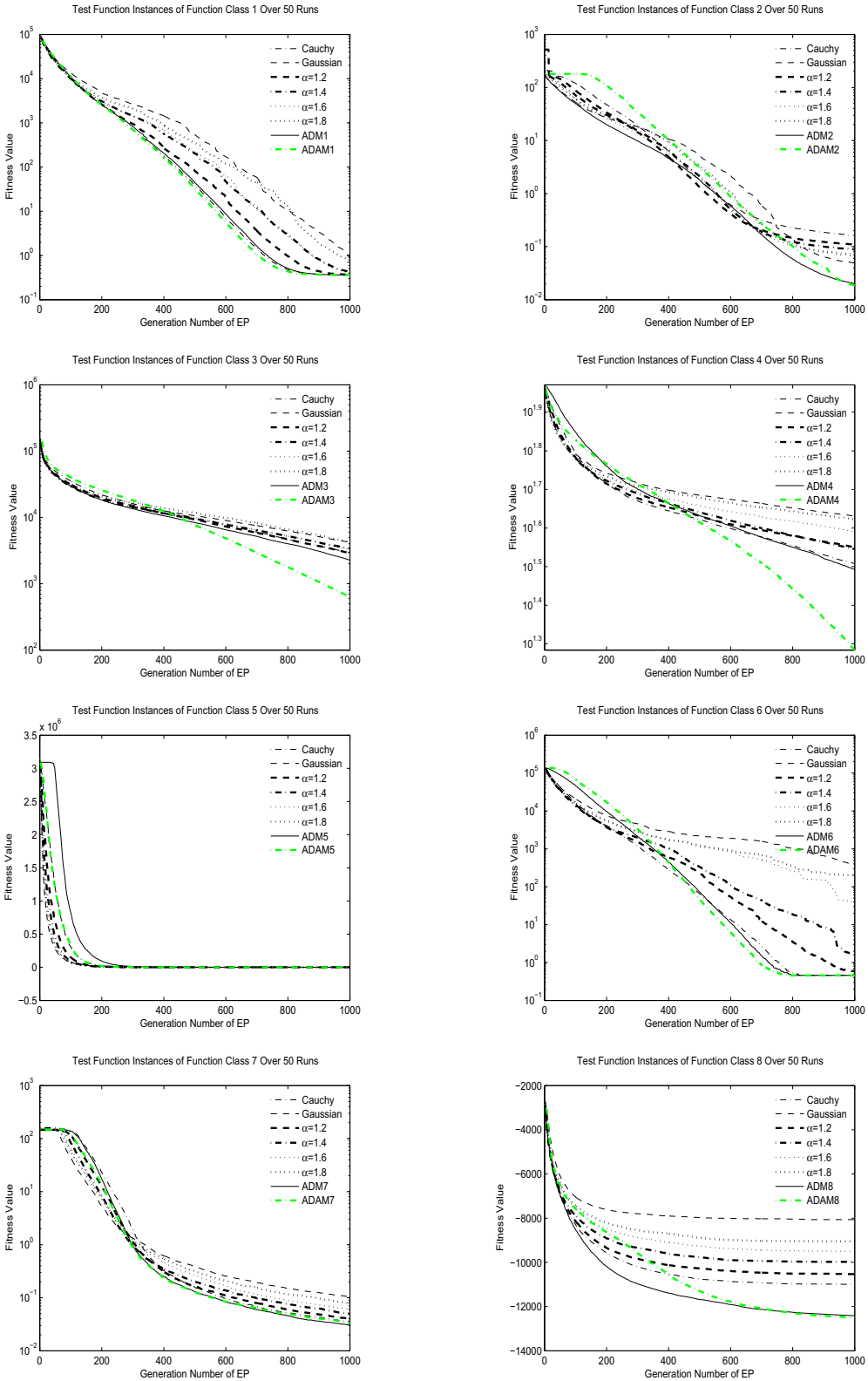

Fig. 2 EP evolutionary process for ADAM, ADM, Gaussian, Cauchy, and Lévy distributions with different values of α. The X-axis represents EP generation, and the Y-axis represents the fitness value of EP

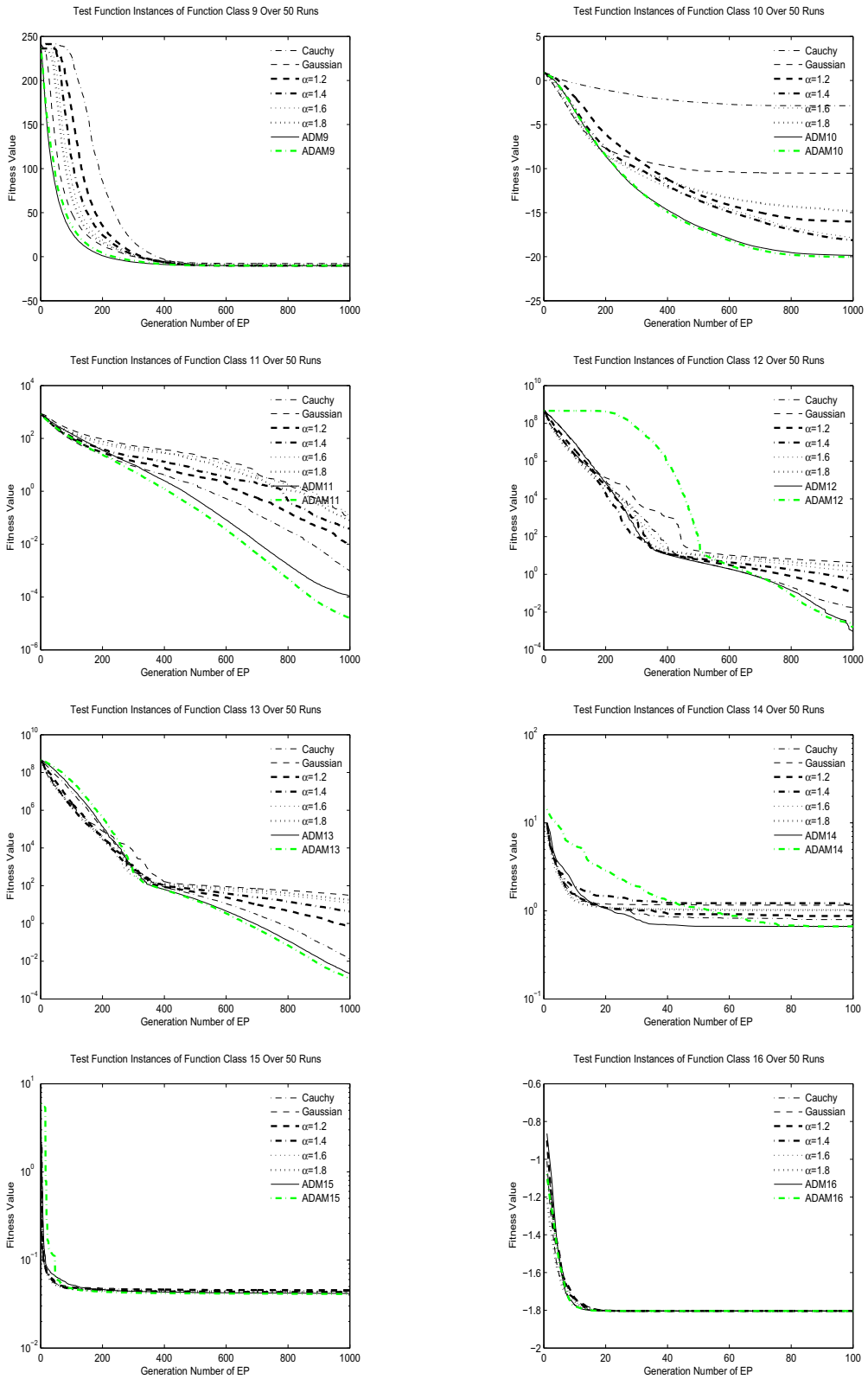

Fig. 3 EP evolutionary process for ADAM, ADM, Gaussian, Cauchy, and Lévy distributions with different values of α. The X-axis represents EP generation, and the Y-axis represents the fitness value of EP

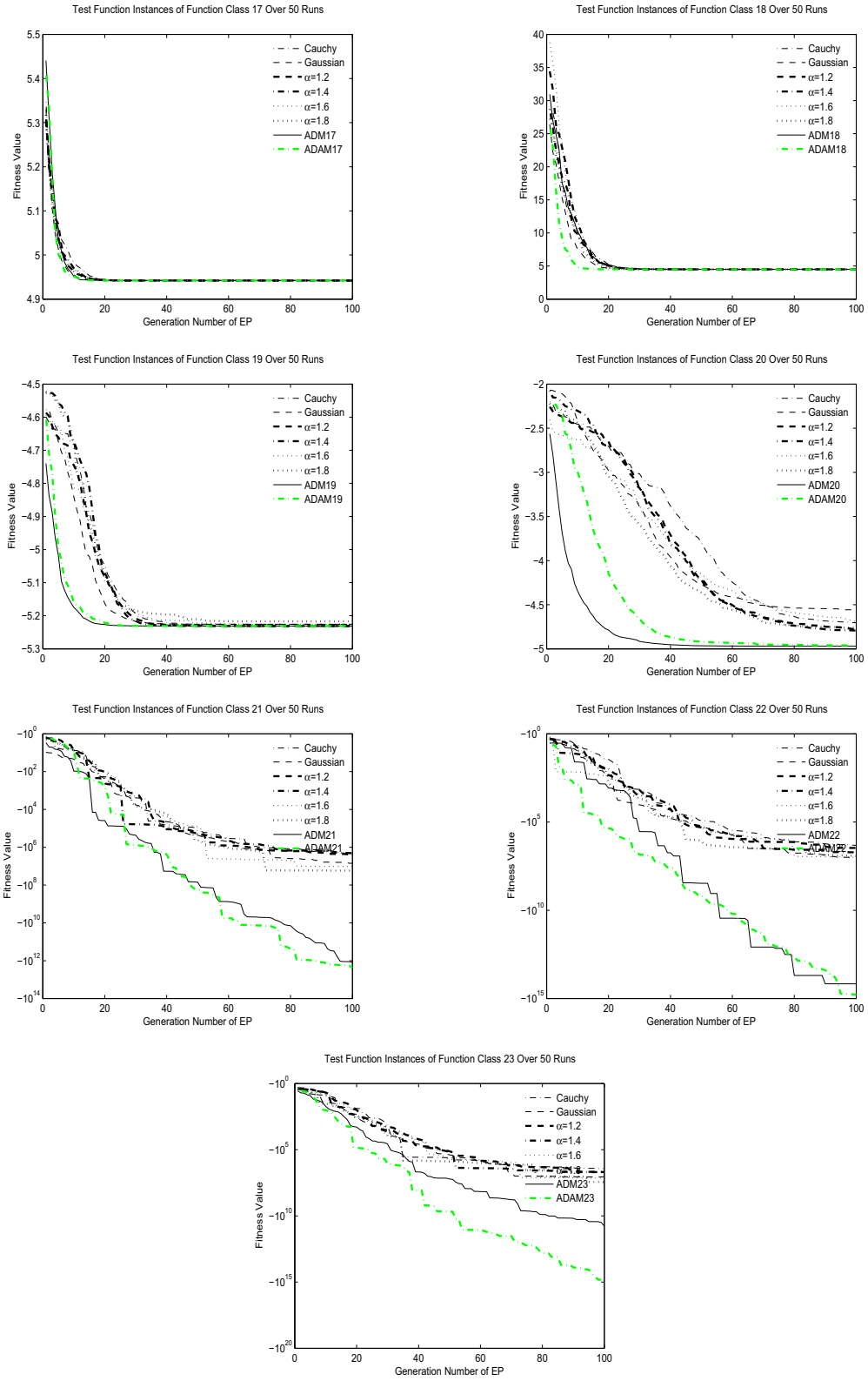

Fig. 4 EP evolutionary process for ADAM, ADM, Gaussian, Cauchy, and Lévy distributions with various values of α. The X-axis represents the EP generation, and the Y-axis represents the fitness value of EP

Table 13 Absolute minimum, absolute maximum, and mean absolute values generated by ADAMs for generations 1, 250, 500, 1000

Gen		$ADAM_1$	$ADAM_2$	$ADAM_3$	$ADAM_4$	$ADAM_5$	$ADAM_6$	$ADAM_7$	$ADAM_8$	$ADAM_9$	$ADAM_{10}$	$ADAM_{11}$	$ADAM_{12}$	$ADAM_{13}$	$ADAM_{15}$
1	Min	0.00198787	0.01227210	0.00000002	0.00000305	0.000551673	0.00402959	0.27649837	2.03815796	0.00007454	0.00038762	0.64790848	0.00679409	0.05090194	3.80E-10
	Max	4916.5	694.5	0.00560605	33.4	3130.1	5319.3	2295.0	2564662.5	173.9	3046.6	32364.5	188949650.4	18593.4	1.21E+07
	Mean	7.00	145.2	0.00046046	0.07489707	6.71	20.1	7.68	1955.9	0.38309484	2.65	27.5	66827.7	23.6	4.57E+03
250	Min	0.00180700	0.00381895	0.00259801	0.00220628	0.00012556	0.00288601	0.25506843	0.00367369	0.00013082	0.00014933	0.95483571	0.03252271	0.05693257	7.95E-17
	Max	4876.7	141.2	459.9	4634.7	11335.9	14515.5	10298.6	4474826.0	327.7	1259.3	20812.6	851359.9	7116.1	7.78E+13
	Mean	7.92	33.9	40.3	20.5	12.2	89.8	10.3	2665.0	1.55	2.93	27.2	3668.9	10.5	4.96E+10
500	Min	0.00114287	0.00021612	0.008554837	0.00727397	0.00041255	0.01495335	0.27797242	0.05869184	0.00000043	0.00021815	1.97396773	0.00304662	0.04360705	8.20E-19
	Max	5435.2	7.17	849.0	18114.9	1223.9	78860.4	991.2	3158493.6	24.1	1116.3	63686.0	85483.8	29230.1	8.97E+12
	Mean	7.10	1.72	72.4	58.6	3.88	88.4	6.09	2265.1	0.13	1.97	60.1	202.4	26.1	3.28E+09
1000	Min	0.00025605	0.00000819	0.000303834	0.00076310	0.00094695	0.00443321	0.26579799	0.17502223	0.00001377	0.00024871	0.55377622	0.00000375	0.00323806	1.48E-18
	Max	2564.4	0.07	2465.1	111169.9	3508.1	239791.3	601.0	1686383.9	243.6	880.1	4905.3	189.7	11.8	4.29E+14
	Mean	5.03	0.02	158.9	179.7	4.57	119.0	5.27	2787.1	0.27	2.31	14.2	0.57	0.08	1.48E+11

For further discussion, consider two components $x^g(\mathrm{j})$ from an individual, $g = 1, 2$, as below:

$$x^g(1) = x^{g-1}(1) + \beta Y_1^{g-1}, \tag{3}$$

$$x^g(2) = x^{g-1}(2) + \beta Y_2^{g-1}. \tag{4}$$

Then, let us use ADAM to generate variants for Y_j^k; we plot the variables $x^g(1)$ and $x^g(2)$ in Figs. 5, 6 and 7 for each ADAM with four different generations: 1, 250, 500, 1000, or alternatively 1, 25, 50, 100. The figures demonstrate that the size of the jumps the mutation operator can make in different EP generations are different. For example, in the early EP generations, the ADAMs search a wider space and can be seen as globally searching the space. In later EP generations, the ADAMs search a very small space. The ADAMs search widely different spaces in the early and later EP generations. This feature leads to the outperformance of ADAMs over the ADMs and human designed mutation operators.

We use ADAMs to generate 3000 random values for different EP generations. The minimum, maximum, and mean values are given in Tables 13 and 14. The values generated by the ADAMs are widely different across generations. In most cases, the minimum and mean values generated by the ADAMs are much smaller for large generation indices, and thus ADAMs search a wider space in the early EP generations, and search smaller space in the later EP generations.

Conclusions and future work

One of the advantages of the proposed method is that it passes the responsibility of the design process, to discover ADAMs for EP, from the human researcher to GP. Different from [7–9], the mutation operators discovered by the method proposed in this paper is 'automatically' designed and 'adaptive'. The GP sits at the hyper-level to generate heuristics as ADAMs for EP on function classes. In other words, it tailors an ADAM to a function class. In this work, we compare two types of automatically designed mutation operators (ADAMs with adaptive factors and ADMs without adaptive factors) as well as human designed ADMs. Adaptive factors may capture the current status and present positions of individuals, or the landscape of the search space. The mutation operators with these factors enable EP to search in a targeted way at different stages. One of the disadvantages of the framework is, due to EP algorithm at the base level, the computational cost needed to evolve both ADAMs and ADMs. However, this is a singular up-front cost, and the trade-off as to whether this is acceptable depends on the application, required solution quality, and the number of times we are expected to solve problems from the target class of problems. A second disadvantage is the need for a number of training problems.

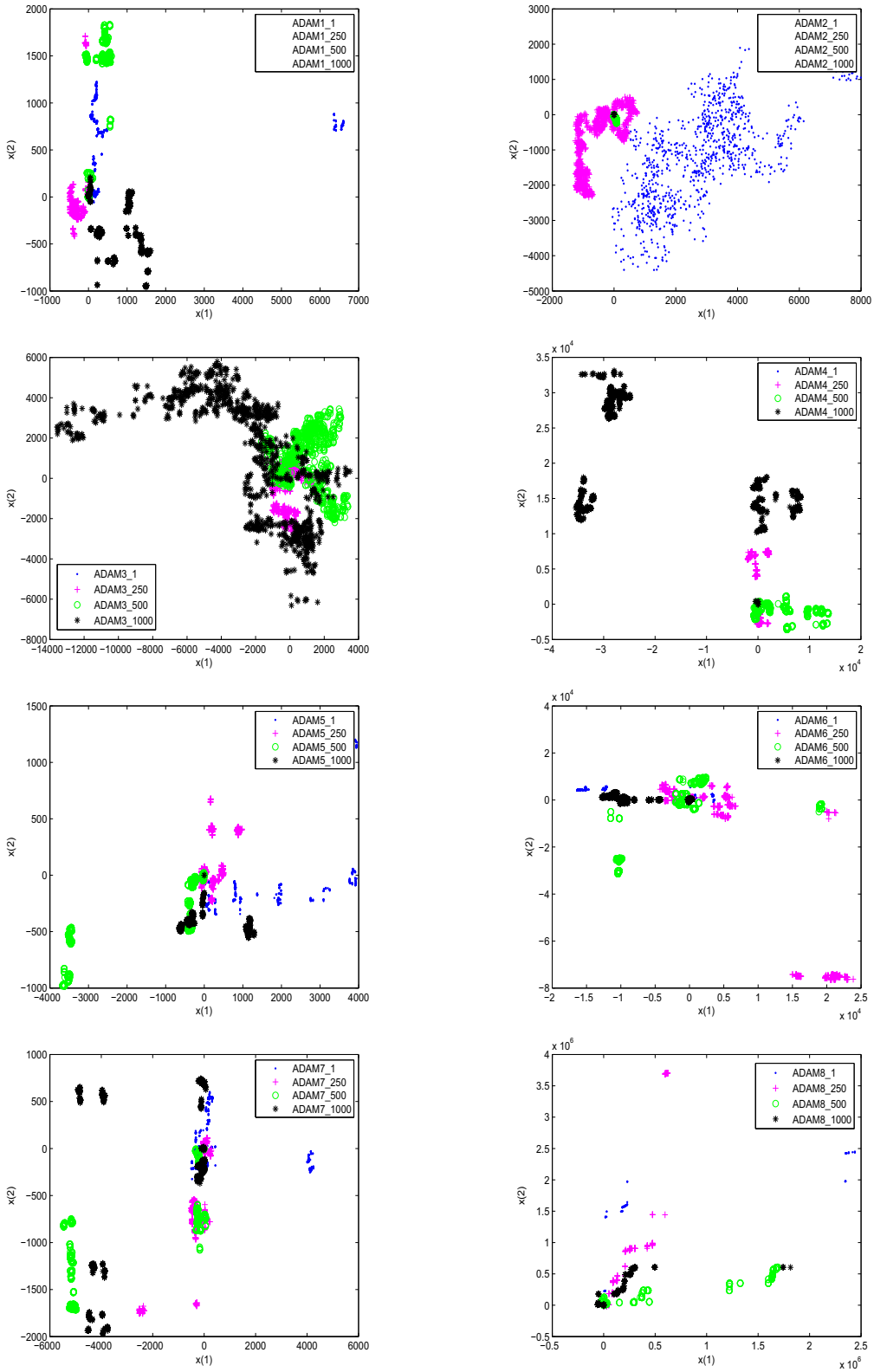

Fig. 5 Plot of the 2-dimensional variables from ADAMs in various generations (1, 250, 500, 1000). The original components are set to $x^0(1) = x^0(2) = 0$ and plotting 1000 with $\beta = 1$

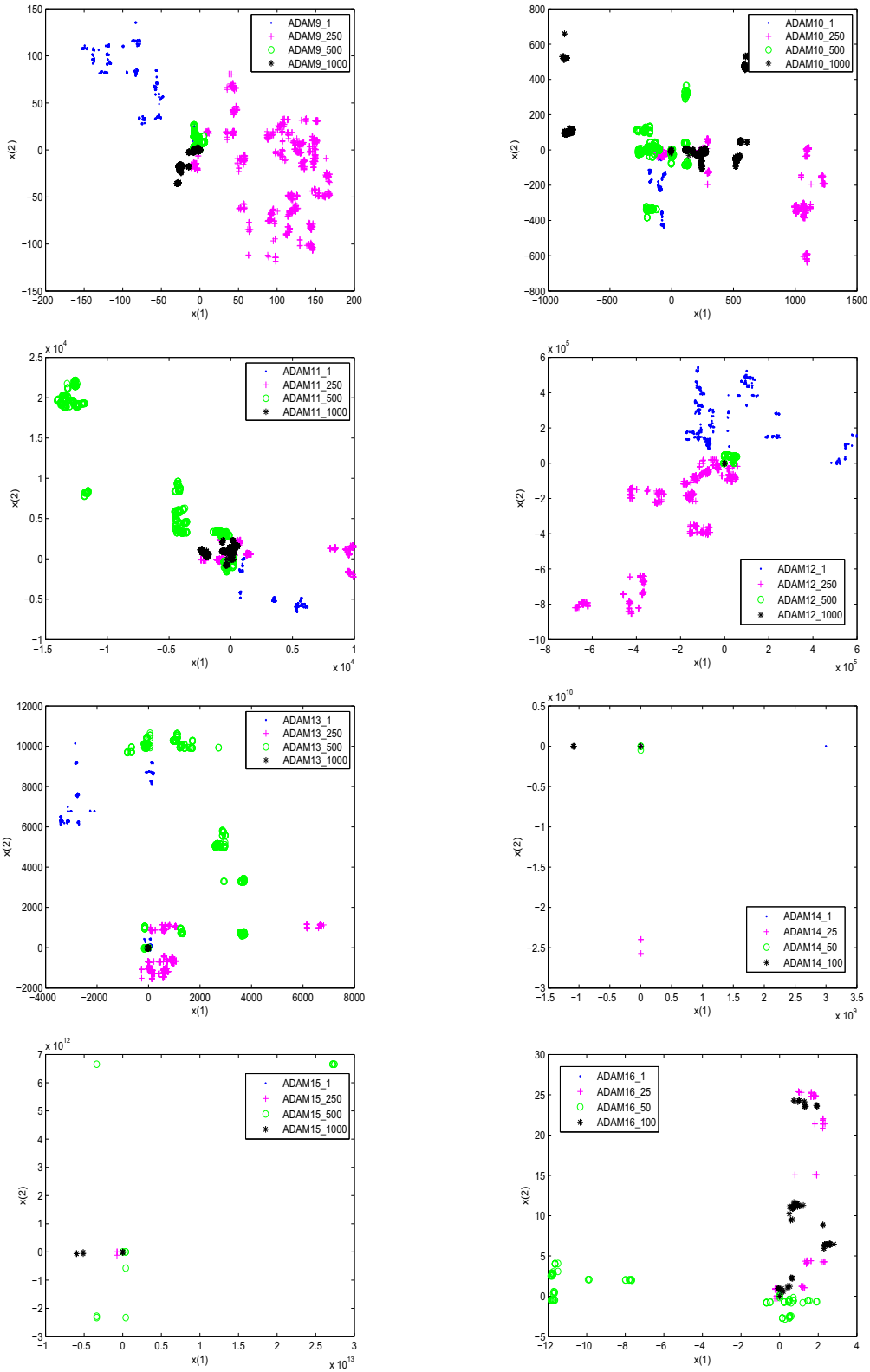

Fig. 6 Plot of the 2-dimensional variables from ADAMs in various generations. The original components are set to $x^0(1) = x^0(2) = 0$ and plotting 1000 or 100 with $\beta = 1$

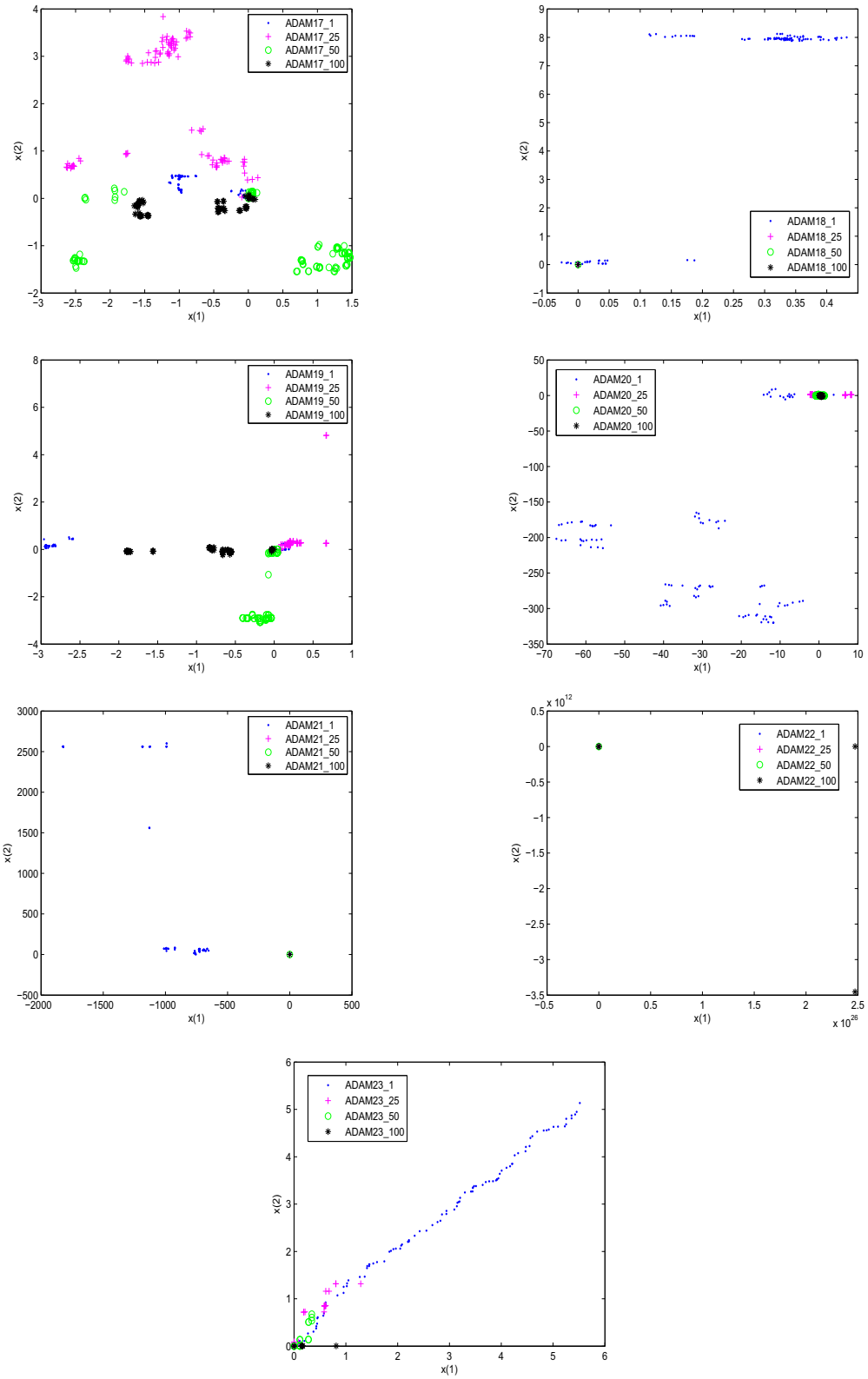

Fig. 7 Plot of the 2-dimensional variables from ADAMs in various generations (1, 25, 50, 100). The original components are set to $x^0(1) = x^0(2) = 0$ and plotting 100 with $\beta = 1$

Table 14 Absolute minimum, absolute maximum, and mean absolute values generated by ADAMs in generations 1, 25, 50, 100

Gen		$ADAM_{14}$	$ADAM_{16}$	$ADAM_{17}$	$ADAM_{18}$	$ADAM_{19}$	$ADAM_{20}$	$ADAM_{21}$	$ADAM_{22}$	$ADAM_{23}$
1	Min	1.17E−09	7.81E−08	2.37E−06	1.95E−03	1.51E−07	2.19E−01	1.68E−02	0.00E+00	6.44E−12
	Max	3.96E+09	1.56E+00	1.49E+01	8.99E+01	2.94E+01	1.45E+03	1.44E+04	2.91E+00	2.35E−01
	Mean	3.78E+06	1.91E−03	2.60E−02	8.70E−02	2.81E−02	4.75E+00	3.43E+01	2.45E−02	5.91E−02
25	Min	1.01E−08	1.00E−09	7.14E−06	3.57E−07	5.31E−07	9.55E−03	2.61E−08	0.00E+00	0.00E+00
	Max	6.71E+10	1.15E+02	6.11E+01	4.45E−02	5.31E+01	5.03E+02	2.36E+01	2.97E+08	6.40E−01
	Mean	3.27E+07	3.60E−01	1.70E−01	2.62E−05	4.75E−02	5.37E−01	8.18E−03	1.32E+05	1.19E−02
50	Min	6.96E−10	3.39E−10	2.36E−05	1.36E−10	2.32E−07	5.53E−03	1.87E−09	0.00E+00	0.00E+00
	Max	2.36E+11	1.47E+02	5.93E+01	1.65E−05	1.47E+02	3.86E+01	2.79E−03	3.04E+15	6.12E−01
	Mean	7.92E+07	4.21E−01	1.09E−01	9.32E−09	8.77E−02	1.54E−01	1.05E−05	1.07E+12	4.96E−03
100	Min	7.73E−09	4.88E−10	1.15E−05	1.40E−10	1.02E−06	2.65E−03	1.01E−10	0.00E+00	0.00E+00
	Max	9.91E+08	5.11E+02	3.36E+01	7.99E−06	3.46E+00	3.16E+01	1.90E−03	1.44E+31	5.24E−01
	Mean	5.40E+05	4.65E−01	4.76E−02	5.49E−09	2.30E−02	7.10E−02	2.43E−06	6.80E+27	3.03E−03

In optimisation, an optimisation algorithm is designed and applied to a single instance of a problem, and that is the entire optimisation process. In our case of automatic design, we are employing a machine learning approach which requires a separate set of training and testing instances. In general, this will require a fair number of training instances, which may not be available.

A large variation in a single mutation enables the EP to globally search a wider region of the search space [14]. Researchers attempted to design mutation strategies that can generate a large variation in the early generations and a small variation in the later EP generations, or small variations in early generations and large variations in the later EP generations. The ADAM can perform well on functions generated from function classes. In Figs. 2, 3 and 4, $ADAM_2$, $ADAM_6$, $ADAM_{13}$, and $ADAM_{23}$ significantly underperform in the early EP generations, as the ADAMs are not well matched with the functions in those early EP generations.

However, their convergence is faster in the later generations due to the adaptive factors, which demonstrate great effectiveness in EP. The experimental data also demonstrate that the adaptive factors collected are useful, and guide the mutation operator in its search from wider regions to small regions for EP. In the early EP generations, according to Figs. 5, 6 and 7, Table 13 and 14, the behaviour of an ADAM is as follows: (1) ADAM searches a wider space and can be seen as searching the space globally in early generations, and then in the later generations it searches a very small space. (2) Alternatively, it searches a very small space earlier and a wider space later, as do $ADAM_3$, $ADAM_4$, and $ADAM_{22}$. The first behaviour is expected, however, the second is more interesting. The search from narrow spaces to wider spaces is due to the change of mutation operators. This means even if the search falls into local optima, the ADAM still has the ability to jump out from the local optima.

In summary, ADAMs search regions vary widely between early and later EP generations, and this feature leads to the superior performance of ADAMs over ADMs and human designed mutation operators. The random values generated by the ADAMs vary significantly across generations: In most cases, the minimum and mean values generated by ADAMs are much smaller for large generation numbers, and thus, the ADAMs search wider spaces in the early EP generations, and smaller spaces in later generations.

In this study, we use an offline hyper-heuristic to automatically design ADAMs for specific function classes. Previously, researchers have used offline hyper-heuristics to tailor ADMs [8] for specific function classes. We conducted tests to evaluate the performance of the tailored ADAMs, tailored ADMs, and mutation operators designed by humans, on specific function classes. The primary conclusions of this study are as follows. First, a tailored ADAM designed by GP can make jumps in the search space of different sizes depending on the EP generation. Secondly, the tailored ADAMs outperformed the tailored ADMs, and the mutation operators designed by humans on most function classes, or achieved the second best performance on a few function classes. Thirdly, an ADAM tailored to functions class performs well on that class of functions when compared to the performance of other ADAMs which were trained on different function classes.

In the future, there are several possible directions that can be investigated. At this point, mutation operators are adaptive, the relationships among adaptive factors are not considered and designed by GP. For example, whether the minimum value and standard deviation exists with certain connections at different generations. It is possible to improve the framework and parameters to further automatically design self-adaptive mutation operators. From the experimental tests, some ADAMs do not demonstrate outstanding performance, which means current adaptive factors

may not well fit the function classes, thus more adaptive factors could be incorporated and their effects investigated. Due to the successful application of the work we present in this study, the idea could also be applied to other meta-heuristics including particle metropolis-hastings [39,40], water wave optimisation [41], differential evolution, and particle swarm optimisation.

Declarations

Conflict of interest On behalf of all the authors, Fuchang Liu states that there is no conflict of interest.

References

1. Poli R, Langdon WB, McPhee NF (2008) A field guide to genetic programming. Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk, 2008. (With contributions by J. R. Koza)
2. Cowling P, Kendall G, Soubeiga E (2000) A hyper-heuristic approach to scheduling a sales summit. In: Practice and theory of automated timetabling. In III: third international conference, PATAT 2000. LNCS. Springer, 2000
3. Dowsland KA, Soubeiga E, Burke E (2007) A simulated annealing based hyperheuristic for determining shipper sizes for storage and transportation. Eur J Oper Res 179(3):759–774
4. Gabriela O, James W, Matthew H, Tim C (2012) Adaptive evolutionary algorithms and extensions to the hyflex hyper-heuristic framework. In parallel problem solving from nature—PPSN XII, volume 7492 of lecture notes in computer science. Springer, Berlin, pp 418–427
5. Pisinger D, Ropke S (2007) A general heuristic for vehicle routing problems. Comput Oper Res 34(8):2403–2435
6. Shao L, Liu L, Li X (2014) Feature learning for image classification via multiobjective genetic programming. IEEE Trans Neural Netwo Learn Syst 25(7):1359–1371
7. Libin H, Drake JH, Woodward JR, Ender Ö (2018) A hyper-heuristic approach to automated generation of mutation operators for evolutionary programming. Appl Soft Comput 20:162–175
8. Libin H, John W, Jingpeng L, Ender Ö (2013) Automated design of probability distributions as mutation operators for evolutionary programming using genetic programming. . Genetic programming, volume 7831 of lecture notes in computer science. Springer, Berlin, pp 85–96
9. Hong L, Drake JH, Woodward JR, Özcan E (2016) Automatically designing more general mutation operators of evolutionary programming for groups of function classes using a hyper-heuristic. In: Proceedings of the genetic and evolutionary computation conference 2016, GECCO '16, New York, NY, USA, pp 725–732, 2016. ACM
10. Ross P, Schulenburg S, Marín-Blázquez JG, Hart E (2002) Hyper-heuristics: learning to combine simple heuristics in bin-packing problems. In: Proceedings of the genetic and evolutionary computation Conference, GECCO '02, San Francisco, CA, USA, pp 942–948, 2002. Morgan Kaufmann Publishers Inc
11. Yao X, Liu Y, Lin G (1999) Evolutionary programming made faster. IEEE Trans Evol Comput 3:82–102
12. Hong L, Drake JH, Özcan E (2014) A step size based self-adaptive mutation operator for evolutionary programming. In: Proceedings of genetic and evolutionary computation conference 2014. ACM, pp 1381–1388, 2014
13. Dong H, He J, Huang H, Hou W (2007) Evolutionary programming using a mixed mutation strategy. Inf Sci 177:312–327
14. Lee C-Y, Yao X (2004) Evolutionary programming using mutations based on the lévy probability distribution. IEEE Trans Evol Comput 8:1–13
15. Mallipeddi R, Suganthan PN (2008) Evaluation of novel adaptive evolutionary programming on four constraint handling techniques. In: Proceedings of IEEE world congress on computational intelligence, pp 4045–4052
16. Mallipeddi R, Mallipeddi S, Suganthan PN (2010) Ensemble strategies with adaptive evolutionary programming. Inf Sci 180(9):1571–1581
17. Shen , He J (2010) A mixed strategy for evolutionary programming based on local fitness landscape. In: 2010 IEEE congress on evolutionary computation (CEC), pp 1–8, July 2010
18. Burke EK, Hyde M, Kendall G, Ochoa G, Özcan E, Woodward JR (2010) A classification of hyper-heuristic approaches. Handbook of metaheuristics. Springer, New York, pp 449–468
19. Chellapilla K (1998) Combining mutation operators in evolutionary programming. IEEE Trans Evol Comput 2(3):91–96
20. Liang KH, Yao X, Liu Y, Newton C, Hoffman D (1998) An experimental investigation of self-adaptation in evolutionary programming, pp 291–300
21. Liu Y (2007) Operator adaptation in evolutionary programming. In: International conference on advances in computation and intelligence. Springer, Berlin, pp 90–99
22. Dong H, He J, Huang H, Hou W (2007) Evolutionary programming using a mixed mutation strategy. Inf Sci 177(1):312–327
23. Yao X, Liu Y (1996) Fast evolutionary programming. In: Proceedings of the 5th annual conference on evolutionary programming. MIT Press, pp 451–460
24. Bäck T, Schwefel H-P (1993) An overview of evolutionary algorithms for parameter optimization. Evol Comput 1:1–23
25. Vinicius Gandra Martins Santos and Marco Antonio Moreira de Carvalho. Adaptive large neighborhood search applied to the design of electronic circuits. Applied Soft Computing, 73:14–23, 2018
26. Zakian P, Kaveh A (2018) Economic dispatch of power systems using an adaptive charged system search algorithm. Appl Soft Comput 73:607–622
27. Al-Sharman MK, Emran BJ, Jaradat MA, Najjaran H, Al-Husari R, Zweiri Y (2018) Precision landing using an adaptive fuzzy multi-sensor data fusion architecture. Appl Soft Comput 69:149–164
28. Wang L, Pei J, Wen Y, Pi J, Fei M, Pardalos PM (2018) An improved adaptive human learning algorithm for engineering optimization. Appl Soft Comput 71:894–904
29. Mashwani WK, Salhi A, Yeniay O, Jan MA, Khanum RA (2017) Hybrid adaptive evolutionary algorithm based on decomposition. Appl Soft Comput 57:363–378

30. Cai Y, Sun G, Wang T, Tian H, Chen Y, Wang J (2017) Neighborhood-adaptive differential evolution for global numerical optimization. Appl Soft Comput 59:659–706
31. Aziz NAA, Ibrahim Z, Mubin M, Nawawi SW, Mohamad MS (2018) Improving particle swarm optimization via adaptive switching asynchronous-synchronous update. Appl Soft Comput 72:298–311
32. Wang G, Tan Y (2019) Improving metaheuristic algorithms with information feedback models. IEEE Trans Cybern 49(2):542–555
33. Li C, Yang S, Korejo I (2008) An adaptive mutation operator for particle swarm optimization. In: Proceedings of the 2008 UK workshop on computational intelligence, pp 165–170
34. Khurana M, Massey K (2015) Swarm algorithm with adaptive mutation for airfoil aerodynamic design. Swarm Evol Comput 20:1–13
35. Sk I, Minhazul DS, Saurav G, Subhrajit R, Suganthan Ponnuthurai N (2012) An adaptive differential evolution algorithm with novel mutation and crossover strategies for global numerical optimization. IEEE Trans Syst Man Cybern Part B (Cybern) 42(2):482–500
36. Luke S, Panait L (2002) Lexicographic parsimony pressure. In: Proceedings of genetic and evolutionary computation conference 2002. Morgan Kaufmann Publishers, pp 829–836
37. Silva S (2007) A genetic programming toolbox for MATLAB. University of Coimbra, Portugal. http://gplab.sourceforge.net/download.html
38. Silva S, Almeida J (2003) Gplab—a genetic programming toolbox for matlab. In: Proceedings of the Nordic MATLAB conference 2003, pp 273–278
39. Martino L, Elvira V, Camps-Valls G (2018) Distributed particle metropolis-hastings schemes. pp 553–557, May 2018
40. Martino L, Elvira V, Camps-Valls G (2018) Group importance sampling for particle filtering and MCMC. Digit Signal Process 82:133–151
41. Zhang J, Zhou Y, Luo Q (2018) An improved sine cosine water wave optimization algorithm for global optimization. J Intell Fuzzy Syst 34(4):2129–2141