Applied Intelligence (2020) 50:3961–3975
https://doi.org/10.1007/s10489-020-01745-w

An adaptive GP-based memetic algorithm for symbolic regression

Jiayu Liang[1] · **Yu Xue**[2]

Published online: 6 July 2020

Abstract
Symbolic regression is a process to find a mathematical expression that represents the relationship between a set of explanatory variables and a measured variable. It has become a best-known problem for GP (genetic programming), as GP can use the tree representation to represent solutions as expression trees. Since the success of memetic algorithms (MAs (Memetic algorithms (MAs) can be regarded as a class of methods that combine population-based global search and local search [6, 30])) has proved the importance of local search in augmenting the global search ability of GP, GP with local search is investigated to solve symbolic regression tasks in this work. An important design issue of MAs is the balance between the global exploration of GP and the local exploitation, which has a great influence on the performance and efficiency of MAs. This work proposes a GP-based memetic algorithm for symbolic regression, termed as aMeGP (adaptive Memetic GP), which can balance global exploration and local exploitation adaptively. Compared with GP, two improvements are made in aMeGP to invoke and stop local search adaptively during evolution. The proposed aMeGP is compared with GP-based and nonGP-based symbolic regression methods on both benchmark test functions and real-world applications. The results show that aMeGP is generally better than both GP-based and nonGP-based reference methods with its evolved solutions achieving lower root mean square error (RMSE) for most test cases. Moreover, aMeGP outperforms the reference GP-based methods in the convergence ability, which can converge to lower RMSE values with faster or similar speeds.

Keywords Adaptive memetic algorithm · Genetic programming · Local search · Crossover · Mutation

1 Introduction

Symbolic regression is to search the space of mathematical expressions to find a model that best fits a given dataset in accuracy and simplicity [21]. As one of evolutionary algorithms (EAs), genetic programming (GP) solves optimization problems by imitating the evolution procedure in nature to evolve computer programs for given tasks, which is expected to find global optima [7]. Since tree-based GP can represent solutions as expression trees, symbolic regression becomes one of the best-known application domains for GP [21]. GP and its variants in the existing GP-based works [16, 17, 21, 31] show that GP is well-performing for symbolic regression tasks. Since GP acts at the syntactic level, a small syntactic modification in a GP solution can produce a dramatic change in its fitness, which can harm search efficiency [28]. To address this issue, the integration of local search into GP, known as memetic algorithms, has attracted significant attention, which can further improve the search ability of GP [25].

Memetic algorithms (MAs) are a combination of population-based evolutionary algorithms (EAs) and individual-based local search [6, 30]. EAs imitate the evolutionary process of natural selection in solving optimization problems, which can conduct global exploration of the solution space; while local search methods exploit neighboring solutions of the solutions evolved by EAs [19]. The success of MAs has proved the importance of local search in augmenting the global search ability of EAs [19]. However, the balance between the global exploration and local exploitation in MAs has a great influence on the performance and efficiency of the MAs [23, 28]. Specifically, applying local search too frequently may lead to expensive computational cost and loss of the exploration capacity [23]. For example,

✉ Jiayu Liang
yyliang2012@hotmail.com

Yu Xue
xueyu@nuist.edu.cn

[1] Tianjin Key Laboratory of Autonomous Intelligent Technology and System, Tiangong University, Tianjin, 300387, China

[2] School of Computer and Software, Nanjing University of Information Science and Technology, Nanjing, 210044, China

Banzhaf et al. apply local improvement on every individual at the current population, which leads to an obvious computational bottleneck [9]. In contrast, EAs without applying local search may experience slow convergence [23].

There are existing works [2, 5, 20, 24, 27] that consider balancing the global and local search in MAs, where not every individual is subject to local search. Two types of common approaches, i.e. random selection and fitness-based selection, are used to select individuals that will undergo local search [19]. Specifically, the works [2, 5, 24, 27] use fitness-based selection, where they apply local search operators for the best individuals of the population at the final generation or at each generation. In addition, Nguyen et al. [20] combine random selection and fitness-based selection. They propose a method that the population is sorted based on fitness and divided into n levels (n is the number of local search operations at each generation). Then one individual per level is randomly selected for local improvement. However, the random selection approach cannot take advantage of the population performance information during evolution; while the best-fitness selection approach is a fixed way, which does not consider the evolutionary situation changes [19]. Moreover, the existing MAs [2, 5, 20, 24, 27] are rarely used to solve symbolic regression tasks, which means that symbolic regression based on MAs is insufficiently investigated.

1.1 Goals

To bridge the gap, this work proposes a GP-based memetic algorithm for symbolic regression, which can balance the global exploration and the local exploitation adaptively. This method is termed as aMeGP (adaptive Memetic GP). Specifically, compared with GP, two improvements are made in aMeGP to invoke and stop local search adaptively during evolution. Firstly, aMeGP introduces a stage to check whether the evolution has degraded or not based on the average fitness of the population. If the evolution is considered to be at a degraded state, the local search will be invoked. Secondly, two adaptive reproduction operators (i.e. adaptive crossover and mutation) are designed for a further check of whether local search should be invoked or stopped based on the performance of their evolved solutions.

The proposed aMeGP will be compared with GP-based methods (standard GP and existing GP-based MAs) and nonGP-based symbolic regression methods. In addition, both benchmark test functions and real-world applications, which are widely used by existing works, are selected for testing the proposed and reference methods. Specifically, we will investigate the following sub-objectives:

- explore whether aMeGP can improve the search ability of GP for the given symbolic regression tasks;
- compare whether aMeGP can outperform existing GP-based and nonGP-based symbolic regression methods;
- investigate further how aMeGP performs during evolution.

The rest of the paper is organized as follows. Section 2 introduces the background, including GP, GP-based symbolic regression works and MAs. Section 3 describes the proposed method, along with the reference methods. In addition, the experiment preparations are presented in Section 4. The results of the proposed and reference methods are described and analyzed in Section 5. Moreover, Section 6 presents further discussions of how the proposed method performs during evolution. Conclusions are drawn in Section 7.

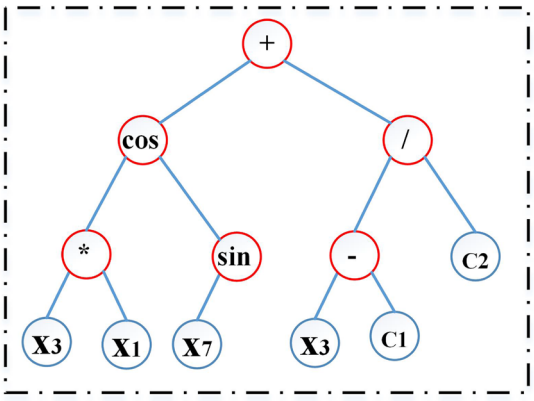
2 Background

This section provides the background information, including the introduction to GP, the existing GP-based symbolic regression methods and the introduction to MAs.

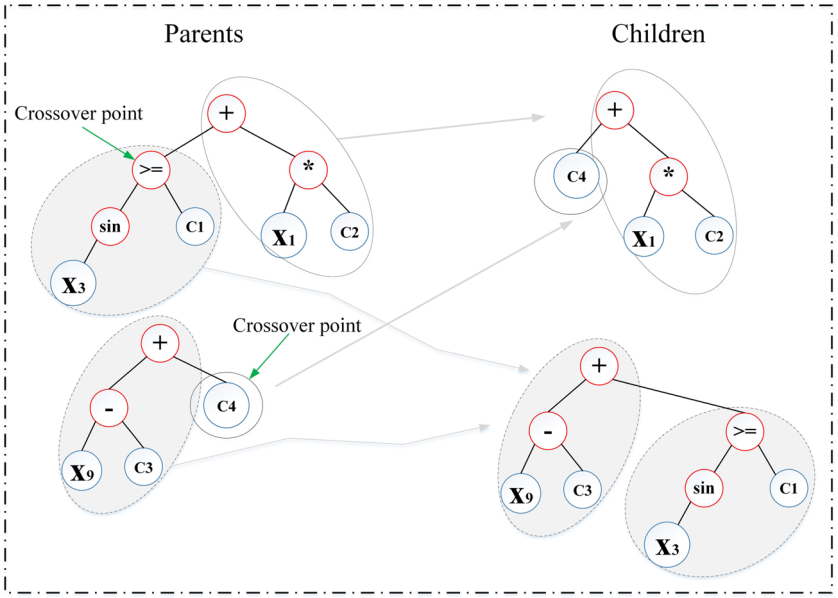
2.1 Genetic programming

Evolutionary algorithms (EAs) are generic population-based meta-heuristic optimization algorithms, and use mechanisms inspired by biological evolution, e.g. reproduction, mutation, recombination and selection [22]. Popularly-used EAs include genetic algorithm, genetic programming (GP), evolutionary strategies, differential evolution and so forth [22]. GP is extended from the basic genetic algorithm, yet its main difference from a genetic algorithm is as follows. GP can represent its solutions in variably-sized structures, e.g. binary string, trees or graphs, which encode the syntax of each individual solution; while GA can only use fixed-sized solution structures [28]. The general framework of GP includes generating an initial population, evaluating the population to assign a fitness to each individual, checking the stop criteria, selecting individuals for reproduction, and generating offspring individuals based on reproduction operators (i.e. crossover, mutation) and elitism (copy the best-performing individuals directly to the next generation).

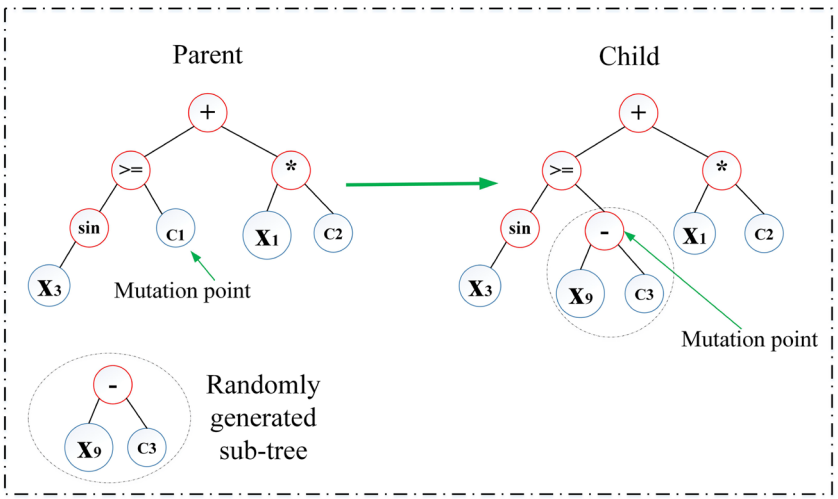
There are three types of GP individual representations, i.e. linear, tree and graph, among which the tree representation (shown in Fig. 1a) is the most widely-used and suitable for symbolic regression tasks [7]. This is because GP with the tree representation can represent solutions as expression trees. In addition, there are two standard reproduction operators in GP, i.e. crossover and mutation [22]. For the crossover operator (shown in Fig. 1b), two parents are required. Specifically, a crossover point is selected in each parent randomly, and then two children are created by replacing the subtrees rooted at the crossover point. For the mutation operator (shown in Fig. 1c), there are also two stages. A mutation point is selected firstly, and then the subtree rooted there is replaced with a subtree generated randomly.



(a) Tree representation



(b) Crossover



(c) Mutation

Fig. 1 The tree representation and the crossover/mutation operators of GP (the terminal set includes nine input variables $x_i, i = 1, 2, ..., 9$ and constants $c_i, i = 1, 2, ...$; the function set includes six operators $+, -, *, /, sin, cos$)

2.1.1 GP for symbolic regression

GP has been shown to be a powerful tool for program induction and automatic modeling. It is well established that GP exhibits good performance on symbolic regression tasks with many existing works [16, 17, 21, 31].

Zhong et al. [31] propose a novel multi-factorial GP (MFGP) algorithm, which can realize evolutionary search on multiple tasks in one independent run. This is the first work that attempts to use a single population to conduct multiple tasks by GP. Pawlak et al. [21] propose a set of semantic-aware initialization operators, selection operators and search operators for GP. The proposed operators are experimentally compared with existing semantic operators for symbolic regression and boolean function synthesis tasks. The results demonstrate that the proposed operators are better in various performance indicators, such as best-of-run fitness and program size. Kronberger et al. [16] investigate the distribution of symbolic regression solutions evolved by GP in the search space. They aim to improve the search for well-fitting solutions based on model similarity that can be pre-computed from a target function. Specifically, they map candidate solutions generated by GP during evolution to the enumerated search space, based on which they find that GP initially explores the whole space and later converges to the subspace of solutions with highest quality. Uriel et al. [17] study the robustness of GP-based methods for symbolic regression. Specifically, they propose a hybrid method based on the RAndom SAmpling Consensus (RANSAC) algorithm and GP, termed as RANSAC-GP, to deal with datasets with outliers. It is the first application of RANSAC to symbolic regression with GP. The results show that the proposed algorithm is able to deal with extreme amounts (90%) of outliers in the training set, which can evolve highly-accurate models.

GP and its variants in the existing GP-based works [16, 17, 21, 31] show that GP is well-performing for symbolic regression tasks. Since GP acts at the syntactic level, a small syntactic modification in a GP solution can produce a dramatic change in its fitness, which can harm search efficiency. To address this issue, the integration of local search into GP, known as memetic algorithms, has attracted significant attention, which can further improve the search ability of GP [25].

2.2 Memetic algorithms

The term "Memetic Algorithms" (MAs) is proposed initially in late 1980s to denote a class of algorithms that blend EAs (Evolutionary Algorithms) with local search methods [19], e.g. simulated annealing [8] and hill-climbing [1]. EAs mimic the evolutionary process of natural selection in solving optimization problems, which can explore the solution space globally. In contrast, local search methods exploit neighboring solutions of certain solutions evolved by EAs and guide the search moving to the one with the locally-best fitness [25].

There are several common design issues to develop a memtic algorithm, including which local search algorithms to select, where to apply local improvement, how to balance the local and global search (e.g. how often the local search will be applied and how long should it last), and so on [19]. Among them, the balance between the global exploration and local exploitation affects the performance and efficiency of the MAs greatly [23, 28]. Specifically, applying local search too frequently may cause expensive computational cost and loss of the exploration capacity; while applying local search too less cannot help to speed up convergence of the global exploration [23]. For example, Banzhaf et al. use local improvement on every individual at the current population, which causes an obvious computational bottleneck [9].

There are existing works that aims to balance the global and local search in MAs [2, 5, 20, 24, 27], where not every individual is subject to local search. To select individuals that will undergo local search, there are two types of most common approaches, i.e. random selection and fitness-based selection [19]. For the fitness-based selection, local search is usually performed on best-performing solutions for local improvement [2]. Specifically, the works [2, 5, 24, 27] use fitness-based selection, where they apply local search operators for the best individuals at the final generation or at each generation of the population. For example, Bansal et al. [5] propose a new local search phase to integrate with the basic artificial bee colony (ABC) to exploit the local search space of the best individual in the swarm. It is aimed to balance between diversity and convergence capability of the ABC. The proposed method is tested on 20 benchmark optimization problem and 4 well-known engineering optimization problems. Results show that new solutions generated locally around the best solution by memetic ABC (MeABC) help to enhance the exploitation capability of basic ABC. In addition, the work [20] combines random selection and fitness-based selection. They propose a method that the population is sorted based on fitness and divided into n levels (n is the number of local search operations at each generation). Then one individual per level is randomly selected for local improvement.

The existing works [2, 5, 20, 24, 27] that aim to balance the global and local search in MAs select individuals randomly or select best individuals in fitness for local improvement. However, the random selection approach cannot take advantage of the evolution information; while the best-fitness selection approach is a fixed way, which does not consider the evolutionary situation changes based on both particular tasks and search stages. Therefore, it is necessary to investigate the MAs that can adaptively balance the global and local search for symbolic regression tasks.

3 Methodology

In this section, the proposed aMeGP (adaptive Memetic Genetic Programming) is described, along with the reference methods.

3.1 The proposed method: aMeGP

The proposed method, aMeGP, can balance the global and local search adaptively, whose pseudo-code is shown in Algorithm 1. Compared with GP (the base technique of aMeGP), aMeGP makes two improvements that are described as follows.

Algorithm 1: Pseudo-code of aMeGP

input : P and N: mean the population and its size respectively;
G: means the maximum number of generations;
The terminal set, function set, GP parameters and other settings (described in Section 4.2).
output: The best solution.

Create an initial population at iteration zero (P_0);
$g \leftarrow 0$;
while $g < G$ *and the ideal solution (the solution with the best fitness) is not found* **do**
  $flag \leftarrow false$;
  Fitness assignment: evaluate each solution on the training set and assign each solution a fitness value;
  Calculate $aveFitP_g$ (the average fitness of P_g);
  if $g > 0$ && $aveFitP_g$ *is worse than* $aveFitP_{g-1}$ **then**
    $flag \leftarrow true$; `// local search is invoked`
  Create P_{g+1} from P_g:
  begin
    Set P_{g+1} empty;
    Copy the best m ($m < N$, decided by the elitism rate) solutions from P_g to P_{g+1};
    if *flag* **then**
      Apply adaptive crossover and mutation;
    else
      Apply standard crossover and mutation;
  $g \leftarrow g + 1$;
Return the best solution of the population at the final generation.

Firstly, it is decided whether the evolution has degraded or not based on the average fitness of the population. If the average population fitness at a generation is worse than that of its previous (parent) generation, the evolution is considered to be at a degraded state and the local search is invoked. In contrast, if the average population fitness at a generation becomes better than that of its previous (parent) generation, the evolution is considered to be at a well-performing state and the local search is not considered at the current generation.

Algorithm 2: Pseudo-code of the adaptive crossover

input : $maxTries$: means the maximum tries for local search;
P_g: means the population at the current generation g;
P_{g+1}: means the population at the next generation $g + 1$.

$n \leftarrow 0$;
$flag \leftarrow false$;
while $n < maxTries$ **do**
  Select two parents (p_1, p_2) by tournament selection from P_g;
  Generate two child solutions (c_1, c_2) from the two parents based on the standard crossover (shown in Fig. 1b);
  if *n==0* **then**
    $bestInd \leftarrow c_1$; `// initialize bestInd`
  if $c_1.fitness$ *better than* $p_1.fitness$ **then**
    Add c_1 to P_{g+1};
    $flag \leftarrow true$;
  else if $c_2.fitness$ *better than* $p_2.fitness$ **then**
    Add c_2 to P_{g+1};
    $flag \leftarrow true$;
  if *flag* **then** `/* find a child individual better than its parent */`
    break;
  else if $c_1.fitness$ *or* $c_2.fitness$ *better than* $bestInd.fitness$ **then**
    $bestInd \leftarrow c_1 or c_2$; `// update bestInd with the best individual generated during local search`
  $n \leftarrow n + 1$;
if $n == maxTries$ **then** `/* no child individual better than its parent founded */`
  Add bestInd to P_{g+1}; `// add the best individual evolved in local search to the next generation`

Secondly, two adaptive reproduction operators (i.e. adaptive crossover and mutation) are designed. They are able to conduct a further check of whether to invoke or stop local search automatically based on the performance of the evolved solutions. Note that local search is introduced into GP by applying it in two reproduction (crossover and mutation) operators, which is a common way and can be easily fitted in a general evolutionary framework of EAs [15]. The proposed adaptive crossover and mutation are shown in Algorithms 2 and 3 respectively. Specifically, the standard reproduction (i.e. crossover and mutation) operators are improved to check whether the evolved offspring solutions perform better than their parents. For a specific reproduction operation (crossover or mutation), if the evolved child solution(s) is/are better than its/their parent, this reproduction operation does not apply local search; otherwise, local search is introduced in this reproduction operation until child solutions better than their parents are found or the max number of local search is reached.

The evolution degradation check and the adaptive reproduction operators are introduced into GP to form the aMeGP, which enable aMeGP to balance local and global search adaptively. Once the evolution is considered to be at a degraded state, the adaptive operators (i.e. crossover and mutation) are invoked until the evolved child solution(s) are better than its/their parent(s) or until the maximum local search number is reached.

Algorithm 3: Pseudo-code of the adaptive mutation

input : $maxTries$: means the maximum tries for local search;
P_g: means the population at the current generation g;
P_{g+1}: means the population at the next generation $g+1$.

$n \leftarrow 0$;

while $n < maxTries$ **do**
  Select one parent (p_1) by tournament selection from P_g;
  Generate one child solution (c_1) from the parent based on the standard mutation (shown in Fig. 1c);
  if n==*0* **then**
    $bestInd \leftarrow c_1$; // initialize bestInd
  if $c_1.fitness$ *better than* $p_1.fitness$ **then**
  /* find a child individual better than its parent */
    Add c_1 to P_{g+1};
    break;
  else if $c_1.fitness$ *better than* $bestInd.fitness$ **then**
    $bestInd \leftarrow c_1$; // update bestInd with the best individual generated during local search
  $n \leftarrow n+1$;
if $n == maxTries$ **then** /* no child individual better than its parent founded */
  Add bestInd to P_{g+1}; // add the best individual evolved in local search to the next generation

3.2 Reference methods

The proposed methods are compared with both GP-based and nonGP-based methods for symbolic regression. Specifically, the three GP-based reference methods include the standard GP and two existing GP-based memetic algorithms. In addition, the nonGP-based reference methods are widely-used symbolic regression methods. Comparison with GP can determine whether the proposed aMeGP can improve the search ability of GP, since the only difference of aMeGP and GP lies in that aMeGP applies local search while GP does not. Comparison with two GP-based memetic methods and four nonGP-based methods can determine whether the proposed aMeGP can outperform existing symbolic regression methods.

3.2.1 GP-based memetic algorithms

The work [4] proposes a local function search operator (LFS) that can search the neighboring region of a given GP solution. Figure 2 presents the search region of the LFS operator. Note that the given function set to build symbolic expressions consists of add (+), subtract (−), multiply (∗), protected divide (/), sine (sin) and cosine (cos). The local search region is defined based on the internal nodes (or functions) of GP tree solutions, where each internal node can be replaced by any swap-compatible functions in the function set. To balance the global and local search, existing works [2, 5, 24, 27] normally select best individuals at each generation or at the final generation for local improvement. Therefore, the two existing memetic algorithms that use the LFS operator to improve the best individuals at the final generation (termed as MA_bestFinal) and at each generation (MA_bestEachGen) are utilized as reference methods.

3.2.2 NonGP-based methods

Four nonGP-based methods are selected as reference, since they are widely used for symbolic regression tasks,

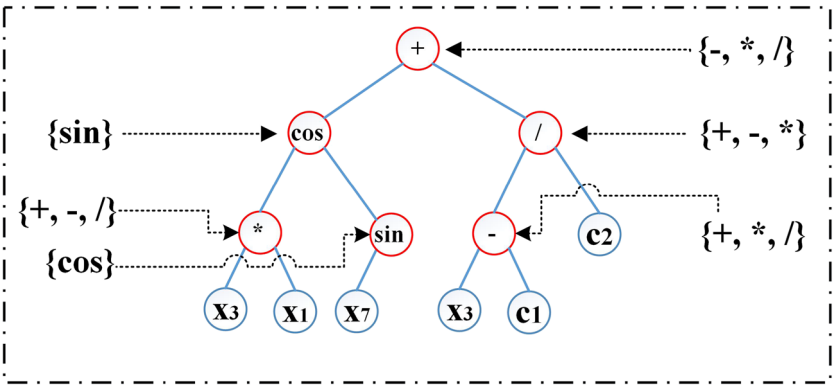


Fig. 2 An example of the LFS (local function search) operator (the terminal set includes nine input variables $x_i, i = 1, 2, ..., 9$ and constants $c_i, i = 1, 2, ...$; the function set includes six operators $+, -, *, /, sin, cos$)

Table 1 Benchmark functions ($U[a, b, c]$ is c uniform random samples drawn from the range $[a, b]$; V.N. means the number of variables)

Name	Function	V.N.	Training/Test data
$Nguyen6(f_1)$	$sin(x_1) + sin(x_1 + x_1^2)$	1	$U[-1, 1, 20]/U[-1, 1, 20]$
$Nguyen7(f_2)$	$log(x_1 + 1) + log(x_1^2 + 1)$	1	$U[0, 2, 20]/U[0, 2, 20]$
$Keijzer4(f_3)$	$x_1^3 e^{-x_1} cosx_1 sinx_1 (sin^2 x_1 cosx_1 - 1)$	1	$E[0, 10, 0.05]/E[0.05, 10.05, 0.05]$
$Sphere5(f_4)$	$x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2$	5	$U[1, 11, 1000]/U[1, 11, 1000]$
$Dic1(f_5)$	$x_1 + x_2 + x_3 + x_4 + x_5$	5	$U[1, 11, 1000]/U[1, 11, 1000]$
$Dic3(f_6)$	$x_1 + \frac{x_2 * x_3}{x_4} + \frac{x_3 * x_4}{x_5}$	5	$U[1, 11, 1000]/U[1, 11, 1000]$
$Nico20(f_7)$	$\sum_{i=1}^{5} (1/x_i)$	5	$U[-5, 5, 1000]/U[-5, 5, 1000]$
$Dic4(f_8)$	$x_1 * x_2 + x_2 * x_3 + x_3 * x_4 * x_5 + x_5 * x_6$	6	$U[1, 11, 1000]/U[1, 11, 1000]$
$Concrete(f_9)$	Concrete compressive strength	8	
$Energy1(f_{10})$	Energy efficiency (Heating Load)	8	
$Energy2(f_{11})$	Energy efficiency (Cooling Load)	8	

Table 2 GP parameter settings

Parameter	Setting	Parameter	Setting
Population Size	500	Generation	50
Initialisation	HalfBuilder	Selection Method	Tournament Selection
Crossover Rate	0.90	Mutation Rate	0.10
Elitism	0.05	Maximum Depth	10

Table 3 Performance ("TrRMSE" and "TeRMSE" refer to the performance in RMSE for training and testing respectively; "size" means the solution size; "TrTime" and "TeTime" stand for the time cost for training and testing respectively; S.T. means significance test; ↑, ↓ and = mean significantly better, worse and similar than/to aMeGP in TeRMSE)

	Measure	GP	MA_BestInd	MA_GenBestInd	aMeGP
f_1	TrRMSE	$9.27E-2$	$6.83E-2$	$7.63E-2$	$4.75E-2$
		$\pm 5.61E-2$	$\pm 6.15E-2$	$\pm 5.24E-2$	$\pm 4.61E-2$
	TeRMSE	$2.28E-1$	$1.75E-1$	$2.03E-1$	$1.92E-1$
		$\pm 1.48E-1$	$\pm 2.34E-1$	$\pm 1.98E-1$	$\pm 3.53E-1$
	S.T.	↓	↑	↓	
	size	82.35 ± 33.90	69.75 ± 30.02	66.75 ± 27.64	76.05 ± 29.05
	TrTime	0.461 ± 0.176	0.597 ± 0.163	0.636 ± 0.152	1.754 ± 1.069
	TeTime	0.0001 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000
f_2	TrRMSE	$1.28E-1$	$1.17E-1$	$1.05E-1$	$4.82E-2$
		$\pm 1.05E-1$	$\pm 9.46E-2$	$\pm 1.02E-1$	$\pm 4.39E-2$
	TeRMSE	$1.96E-1$	$1.59E-1$	$1.44E-1$	$1.17E-1$
		$\pm 2.05E-1$	$\pm 1.24E-1$	$\pm 1.31E-1$	$\pm 1.8E-1$
	S.T.	↓	↓	↓	
	size	55.30 ± 21.40	55.50 ± 21.03	50.05 ± 15.71	65.90 ± 27.22
	TrTime	0.374 ± 0.167	0.479 ± 0.219	0.521 ± 0.217	1.621 ± 0.647
	TeTime	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000
f_3	TrRMSE	$3.03E-1$	$2.96E-1$	$2.94E-1$	$2.73E-1$
		$\pm 1.42E-2$	$\pm 3.18E-2$	$\pm 4.23E-2$	$\pm 4.72E-2$
	TeRMSE	$3.11E-1$	$3.01E-1$	$3E-1$	$2.74E-1$
		$\pm 1.78E-2$	$\pm 2.84E-2$	$\pm 4.24E-2$	$\pm 4.59E-2$
	S.T.	↓	↓	↓	
	size	48.10 ± 34.90	48.50 ± 23.44	56.30 ± 28.99	61.40 ± 42.22
	TrTime	3.972 ± 1.550	3.964 ± 1.426	7.929 ± 3.561	19.968 ± 10.594
	TeTime	0.0004 ± 0.0006	0.0004 ± 0.0002	0.0007 ± 0.0004	0.0003 ± 0.0002
f_4	TrRMSE	$2.42E1$	$2.33E1$	$2.13E1$	$1.95E1$
		$\pm 3.46E0$	$\pm 3.07E0$	$\pm 3.7E0$	$\pm 3.9E0$

including the simple linear regression (SLR), the additive regression (AR), a decision tree based method (REPTree) and a support vector regression method (SMOreg) from the Weka package[1] [11]. Specifically, the SLR aims to learn a simple linear regression model; the AR can learn an additive model (AM), which is a nonparametric regression method; the REPTree is a fast decision tree learner, which can develop a regression tree based on the information gain/variance; the SMOreg is an implementation of sequential minimal optimization (SMO) for support vector regression.

4 Experiment preparation

In this section, the benchmark dataset and GP settings (e.g. the terminal and function sets, the GP parameters and the fitness function) are introduced. In addition, the algorithms in this work are run on the same computer. The processor of this computer is "Intel(R) Core(TM) i5-7200U @ 2.50GHz". Its RAM is 16GB, whose frequency is 2133 MHz.

4.1 Dataset

The proposed methods are tested on both benchmark functions (e.g. Nguyen, Sphere, Dic, Nico functions) and three real-world tasks from two UCI (University of California Irvine) datasets [10] (e.g. the Concrete and Energy datasets). Details are shown in Table 1. These functions are selected since they have been widely used in the existing works [3, 12, 18], and they vary in difficulty levels. Specifically, there are relatively simpler functions, e.g. Nguyen and Keijzer functions that contain one or three variables; while there are also more complex functions, e.g. Sphere, Dic and Nico functions that have 5 or 6 variables. In addition, the Concrete task is from the UCI Concrete

[1] Weka is a set of machine learning algorithms for solving real-world data mining tasks [11].

Table 4 Performance ("TrRMSE" and "TeRMSE" refer to the performance in RMSE for training and testing respectively; "size" means the solution size; "TrTime" and "TeTime" stand for the time cost for training and testing respectively; S.T. means significance test; ↑, ↓ and = mean significantly better, worse and similar than/to aMeGP in TeRMSE)

	Measure	GP	MA_BestInd	MA_GenBestInd	aMeGP
	TeRMSE	$2.48E1$	$2.38E1$	$2.59E1$	$2.03E1$
		$\pm 3.39E0$	$\pm 3.21E0$	$\pm 1.76E1$	$\pm 4.16E0$
	S.T.	↓	↓	↓	
	size	63.70 ± 16.94	65.05 ± 21.48	69.70 ± 24.79	70.10 ± 16.83
	TrTime	19.083 ± 7.947	18.273 ± 5.751	25.438 ± 13.243	69.669 ± 36.907
	TeTime	0.0017 ± 0.0011	0.0013 ± 0.0010	0.0014 ± 0.0008	0.0010 ± 0.0004
f_5	TrRMSE	$7.18E-4$	$7.18E-4$	$7.18E-4$	$7.19E-4$
		$\pm 2.29E-6$	$\pm 1.68E-6$	$\pm 1.64E-6$	$\pm 1.62E-6$
	TeRMSE	$6.96E-4$	$7.01E-4$	$6.94E-4$	$6.94E-4$
		$\pm 2.61E-6$	$\pm 2.77E-5$	$\pm 2.04E-6$	$\pm 8.67E-7$
	S.T.	=	=	=	
	size	33.55 ± 19.23	28.45 ± 12.61	37.40 ± 16.08	22.65 ± 16.12
	TrTime	13.200 ± 3.881	9.439 ± 2.878	14.391 ± 5.715	34.490 ± 27.684
	TeTime	0.0011 ± 0.0006	0.0007 ± 0.0003	0.0012 ± 0.0006	0.0004 ± 0.0003
f_6	TrRMSE	$7.53E0$	$7.06E0$	$6.59E0$	$6.13E0$
		$\pm 1.44E0$	$\pm 1.85E0$	$\pm 1.73E0$	$\pm 2.66E0$
	TeRMSE	$7.22E0$	$6.91E0$	$6.49E0$	$5.93E0$
		$\pm 1.35E0$	$\pm 1.69E0$	$\pm 1.52E0$	$\pm 2.47E0$
	S.T.	↓	↓	↓	
	size	55.60 ± 18.82	57.50 ± 19.79	60.45 ± 18.27	65.90 ± 24.16
	TrTime	18.124 ± 6.897	12.281 ± 5.116	21.395 ± 14.033	71.264 ± 36.333
	TeTime	0.0016 ± 0.0012	0.0011 ± 0.0006	0.0015 ± 0.0013	0.0011 ± 0.0005
f_7	TrRMSE	$6.77E0$	$5.91E0$	$6.7E0$	$4.76E0$
		$\pm 1.78E0$	$\pm 2.21E0$	$\pm 2.6E0$	$\pm 2.35E0$
	TeRMSE	$1.13E1$	$1.04E1$	$1.14E1$	$7.88E0$
		$\pm 2.12E0$	$\pm 4.11E0$	$\pm 8E0$	$\pm 4.16E0$
	S.T.	↓	↓	↓	
	size	41.70 ± 18.75	36.10 ± 13.65	43.35 ± 18.72	51.75 ± 16.15
	TrTime	21.610 ± 7.502	14.674 ± 2.949	16.786 ± 6.886	106.906 ± 28.435
	TeTime	0.0015 ± 0.0010	0.0011 ± 0.0006	0.0014 ± 0.0009	0.0015 ± 0.0007
f_8	TrRMSE	$2.45E1$	$2.1E1$	$2.07E1$	$1.55E1$
		$\pm 1.62E1$	$\pm 1.14E1$	$\pm 1.29E1$	$\pm 9.03E0$
	TeRMSE	$2.55E1$	$2.18E1$	$2.11E1$	$1.59E1$
		$\pm 1.72E1$	$\pm 1.2E1$	$\pm 1.32E1$	$\pm 9.2E0$
	S.T.	↓	↓	↓	
	size	58.60 ± 26.87	65.60 ± 22.93	55.85 ± 18.14	48.20 ± 19.53
	TrTime	17.483 ± 5.515	15.825 ± 5.060	15.765 ± 5.917	52.717 ± 24.703
	TeTime	0.0016 ± 0.0010	0.0015 ± 0.0006	0.0011 ± 0.0004	0.0009 ± 0.0004
f_9	TrRMSE	$1.41E1$	$1.44E1$	$1.35E1$	$1.29E1$
		$\pm 1.01E0$	$\pm 1.1E0$	$\pm 1.79E0$	$\pm 1.73E0$
	TeRMSE	$1.56E1$	$1.91E1$	$1.53E1$	$1.67E1$
		$\pm 1.49E0$	$\pm 1.22E1$	$\pm 2.73E0$	$\pm 7.63E0$
	S.T.	=	↓	↑	
	size	107.55 ± 47.95	104.70 ± 41.73	82.95 ± 29.29	104.95 ± 36.20
	TrTime	29.340 ± 15.919	22.947 ± 10.397	27.424 ± 16.117	122.258 ± 59.459
	TeTime	0.0021 ± 0.0018	0.0013 ± 0.0008	0.0010 ± 0.0006	0.0011 ± 0.0006
f_{10}	TrRMSE	$3.8E0$	$3.65E0$	$3.42E0$	$3.01E0$
		$\pm 4.36E-1$	$\pm 5.98E-1$	$\pm 5.38E-1$	$\pm 4.97E-1$
	TeRMSE	$5.18E0$	$4.7E0$	$4.89E0$	$3.96E0$
		$\pm 1.09E0$	$\pm 9.31E-1$	$\pm 1.13E0$	$\pm 1.03E0$

Table 4 (continued)

	Measure	GP	MA_BestInd	MA_GenBestInd	aMeGP
	S.T.	↓	↓	↓	
	size	64.95 ± 24.91	63.20 ± 22.73	71.65 ± 26.05	83.10 ± 23.68
	TrTime	9.354 ± 4.223	7.458 ± 2.772	8.819 ± 3.524	42.770 ± 17.891
	TeTime	0.0005 ± 0.0003	0.0004 ± 0.0002	0.0004 ± 0.0003	0.0004 ± 0.0002
f_{11}	TrRMSE	$3.57E0$	$3.68E0$	$3.53E0$	$3.11E0$
		$\pm 3.86E-1$	$\pm 3.57E-1$	$\pm 4.63E-1$	$\pm 5.34E-1$
	TeRMSE	$4.46E0$	$4.56E0$	$4.53E0$	$3.69E0$
		$\pm 9.15E-1$	$\pm 5.39E-1$	$\pm 8.8E-1$	$\pm 5.8E-1$
	S.T.	↓	↓	↓	
	size	55.65 ± 21.29	57.55 ± 21.23	62.40 ± 21.30	71.70 ± 33.12
	TrTime	8.801 ± 4.270	7.260 ± 2.758	8.267 ± 4.207	49.825 ± 33.581
	TeTime	0.0005 ± 0.0004	0.0003 ± 0.0001	0.0003 ± 0.0002	0.0004 ± 0.0003

dataset, which aims to generate the concrete compressive strength based on eight-dimensional features (or variables). The Energy1 and Energy2 tasks are both from the UCI Energy dataset. Energy1 is to generate the energy efficiency measured by the heating load; while Energy2 is to generate the energy efficiency measured by the cooling load.

4.2 GP settings

In this part, the terminal set, the function set, the fitness function, and major GP parameters are described. Note that GP develops solutions using the elements from the terminal set and the function set [22]. The function set contains six simple arithmetic operators, i.e. add (+), subtract (-), multiply (*), protected divide (/), sine (sin) and cosine (cos) functions. Specifically, the protected division operator is shown in (1) that considers the case when the divisor is zero. In addition, the terminal set consists of input variables (x_n, n is the number of variables) and constant values that are randomly selected from $[-1, 1]$. The terminal set and the function set are set based on the existing symbolic regression works [12, 18].

$$x/y = \begin{cases} x/y & \text{if y != 0} \\ 0 & \text{if y == 0} \end{cases} \tag{1}$$

A fitness function should be pre-defined to evaluate how well each individual has learned to solve a target problem [22]. root mean square error (RMSE, shown in (2)) is popularly utilized to evaluate symbolic regression tasks [12, 18], which is selected as the fitness function in this work. In addition, major GP parameters are presented in Table 2, which follow the settings of Zhang's work [29], since Zhang decides the suitable GP parameters using experiments for symbolic regression problems in his work. Other default parameters follow the settings of Koza's works [13, 14], who is known for pioneering the use of GP. Moreover, as GP-based methods are not deterministic, each GP-based method run 20 times and the results reported in this work are average values of 20 runs.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}; \tag{2}$$

where N is the number of samples; y_i and $\hat{y}_i$ are the real output and the predicted output of the i sample respectively.

5 Experiment

In this section, the results of the proposed aMeGP are analyzed by comparing with the three GP-based and four nonGP-based reference methods.

5.1 Comparison with GP-based methods

In this part, the proposed aMeGP is compared with three GP-based methods, i.e. GP and two memetic algorithms (termed as MA_BestInd and MA_GenBestInd). Specifically, MA_BestInd applies local search on the best individual of a GP run; while MA_GenBestInd applies local search on the best individuals of each generation during a run. Tables 3 and 4 show the performance of the four methods in training/testing RMSE, the solution size (the number of the node in a GP solution), the training/testing time, and the significance test based on Mann-Whitney U-Test. Note that Mann-Whitney U-Test is often used to determine whether two given independent samples have the same distribution [26], which is used for significance test at the default 5% significance level in this work.

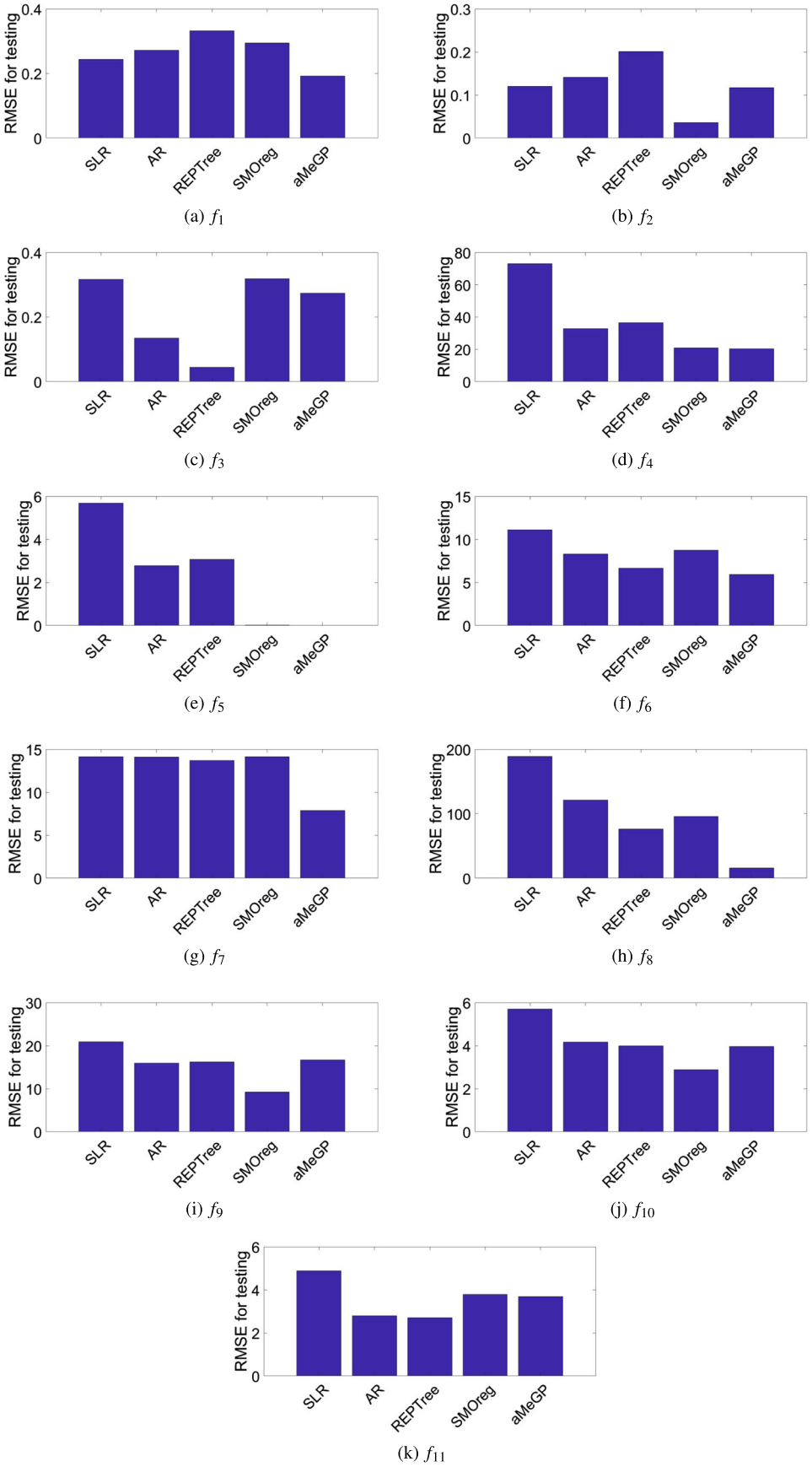
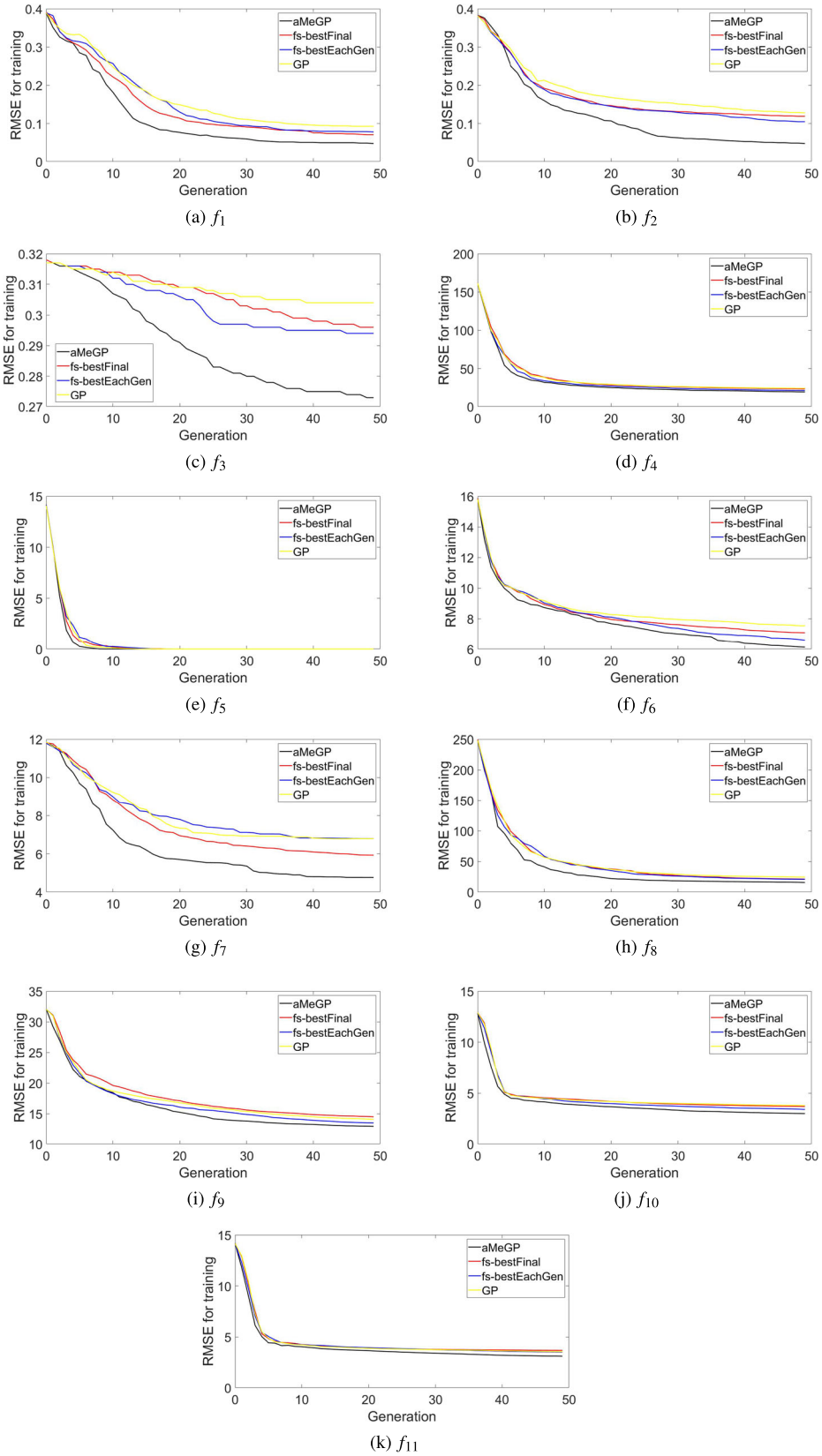


Fig. 3 Comparison of aMeGP with nonGP-based methods in RMSE for testing (the smaller the better)



(a) f_1 (b) f_2

(c) f_3 (d) f_4

(e) f_5 (f) f_6

(g) f_7 (h) f_8

(i) f_9 (j) f_{10}

(k) f_{11}

Fig. 4 The performance in RMSE at each generation during evolution (the smaller the better for RMSE)

In terms of RMSE, the proposed aMeGP outperforms the other three methods for training on all the test functions, except for function f_5. On function f_5, the four methods achieve similar TrRMSE (RMSE for training) values. In addition, aMeGP also performs better than others in TeRMSE (RMSE for testing) generally. Specifically, aMeGP outperforms GP in TeRMSE for 9 out of 11 cases, and outperforms MA_BestInd and MA_GenBestInd for 10 and 9 out of 11 cases respectively. This reflects that the proposed aMeGP is generally better than the reference GP-based methods in symbolic regression. For example, on function f_2, the mean TeRMSE of aMeGP is 1.17E-1; while those of others are 1.96E-1, 1.59E-1 and 1.44E-1. In terms of the solution size, compared with the reference methods, the results are varied that aMeGP solutions can be larger, similar or smaller.

In terms of the time cost, aMeGP consumes more time for training than the three reference methods. It is obvious that aMeGP is more time-consuming than GP, since the difference of aMeGP and GP lies in that aMeGP introduces local search while GP does not, which can increase time cost in aMeGP. In addition, compared with the other two reference methods (i.e. MA_BestInd and MA_GenBestInd) that also involve local search, aMeGP uses more training time. This is because local search of MA_BestInd and MA_GenBestInd is only applied on the best individuals; while local search of aMeGP is invoked adaptively, which involve more individuals. In contrast, compared with GP, MA_BestInd and MA_GenBestInd, aMeGP consumes similar or even less time for testing. This reflects that even though aMeGP needs more time for training, the evolved solutions are efficient.

5.2 Comparison with NonGP-based methods

In this part, the proposed aMeGP is compared with four popularly-used symbolic regression methods, i.e. SLR (simple linear regression), AR (additive regression), REPTree (a decision tree based method) and SMOreg (a support vector regression method).

Figure 3 presents the testing performance of the five methods in RMSE that is the smaller the better. In Fig. 3, aMeGP ranks first on 6 test functions (f_1, f_4, f_5, f_6, f_7 and f_8) out of 11 cases; while it ranks second or performs similar to the second one on 4 test functions (f_2, f_9, f_{10} and f_{11}) out of 11 cases. Even though on the left one function f_3, AR and REPTree are better than the proposed aMeGP, they cannot perform consistently well for most of the given tasks. For example, aMeGP outperforms AR and REPTree on functions f_1, f_2, f_4 and so forth. Therefore, aMeGP performs better than or similar to most reference methods consistently.

6 Further analyses

In this section, the performance of aMeGP during evolution is investigated, along with the three GP-based reference methods. Figure 4 presents the training performance in RMSE of the four methods during evolution. Note that it is the smaller the better for RMSE.

In Fig. 4, it can be seen that aMeGP converges to lower RMSE values than the other three reference methods eventually for most of the given tasks. Specifically, aMeGP achieves lower RMSE values at generation 50 for 9 out of 11 cases (i.e. functions f_1, f_2, f_3, f_6, f_7, f_8,- f_9, f_{10}, f_{11}); while it achieves similar RMSE values for the left two cases (i.e. functions f_4 and f_5). For example, the RMSE performance of aMeGP reaches around 0.05 on function f_2; while that of all the other three methods are all higher than 0.1.

Figure 4 also shows that aMeGP converges faster than or similar to the other reference methods on all the given tasks. Specifically, aMeGP converges faster than others on four functions, i.e. f_1, f_2, f_3, f_7, especially at the first several generations. In other words, the performance curve of aMeGP in Fig. 4 is steeper than those of others, which is obvious in the beginning of evolution. In addition, aMeGP has similar convergence speed to the reference methods on the left test functions.

In summary, the proposed aMeGP performs better than the reference GP-based methods during evolution, as it can converge to lower RMSE values with faster or similar speeds. In other words, the convergence ability of aMeGP is better than GP and two existing MA methods (i.e. fs-bestFinal and fs-bestEachGen).

7 Conclusions

In this work, a new GP-based memetic algorithm for symbolic regression is designed, which can balance global exploration and local exploitation adaptively. This method is termed as aMeGP (adaptive Memetic GP), which takes GP as the base technique and introduces adaptive local search into GP. Specifically, to balance global and local search, aMeGP introduces the evolution degradation check and the adaptive crossover/mutation operators into GP, which helps to invoke and stop local search adaptively during evolution. The proposed aMeGP is compared with GP-based and nonGP-based symbolic regression methods on both benchmark test functions and real-world applications. The results show that aMeGP is generally better than both GP-based and nonGP-based reference methods with its evolved solutions achieving lower RMSE values for most test cases. In addition, even though aMeGP consumes more

training time, its evolved solutions are efficient for testing. Moreover, aMeGP outperforms the GP-based reference methods in the convergence ability, which can converge to lower RMSE values with faster or similar speeds.

Memetic algorithms have achieved success in various areas, e.g. scheduling, routing and combinatorial optimization problems. In real-life applications, multiple conflicting points of view are often taken into account, which leads to multiple objective optimization problems (MOP). Memetic algorithms have been proved to be one of the most efficient algorithms for single objective optimization. Therefore, the attempt of extending memetic algorithms to multi-objective optimization is increasing, which we would like to investigate in the future.

Acknowledgments This study is funded by National Natural Science Foundation of China (grant number 61902281 and 61876089) and Tianjin Science and Technology Program (grant number 19PTZWHZ00020).

Compliance with Ethical Standards This article does not contain any studies with human participants or animals performed by any of the authors.

Conflict of interests Author Jiayu Liang declares that she has no conflict of interest. Author Yu Xue declares that he has no conflict of interest.

References

1. Al-Betar MA, Aljarah I, Awadallah M, Faris H, Mirjalili S (2019) Adaptive β - hill climbing for optimization. Soft Computing 23(1):13489–13512
2. Aleb N, Tamen Z (2011) A memetic algorithm for program verification. In: Proceedings of the UKSim 5th European Symposium on Computer Modeling and Simulation, EMS 2011, Madrid, Spain, November 16-18, 2011
3. Anjum A, Sun F, Wang L, Orchard J (2019) A novel continuous representation of getic programmings using recurrent neural networks for symbolic regression. arXiv:1904.03368
4. Azad RMA, Ryan C (2014) A simple approach to lifetime learning in genetic programming-based symbolic regression. Evol Comput 22(2):1–30
5. Bansal JC, Sharma H, Arya KV, Nagar A (2013) Memetic search in artificial bee colony algorithm. Soft Comput 17(10):1911–1928
6. Boryczka U, Szwarc K (2019) Selected variants of a memetic algorithm for jsp – a comparative study. Int J Prod Res 44:1–16
7. Burlacu B, Affenzeller M, Kommenda M, Kronberger G (2018) Winkler s.: Schema analysis in Tree-Based genetic programming
8. Chopard B, Tomassini M (2018) Simulated annealing. In: An introduction to metaheuristics for optimization. Springer, Cham, pp 59-79
9. Deb K, Poli R, Banzhaf W, Beyer H, Burke E, Darwen P, Dasgupta D (2003) Floreano d.: Genetic and evolutionary computation - GECCO
10. Dua D, Graff C (2017) UCI machine learning repository. http://archive.ics.uci.edu/ml
11. Frank E, Hall MA, Witten IH (2016) Data mining: Practical machine learning tools and techniques (fourth edition)
12. Izadi Rad H, Feng J, Iba H (2018) GP-RVM: Genetic programing-based symbolic regression using relevance vector machine
13. Koza JR (1999) Genetic programming III: Darwinian invention and problem solving, vol 3, Morgan Kaufmann
14. Koza JR, Keane MA, Streeter MJ, Mydlowec W, Yu J, Lanza G (2006) Genetic programming IV: Routine human-competitive machine intelligence, vol 5, Springer Science and Business Media, Berlin
15. Krasnogor N, Hart WE, Smith J, Pelta DA (1999) Protein structure prediction with evolutionary algorithms. Available at http://eprints.uwe.ac.uk/11083
16. Kronberger G, Kammerer L, Burlacu B, Winkler SM, Kommenda M, Affenzeller M (2019) Cluster analysis of a symbolic regression search space. Genetic Programming Theory and Practice XVI 85–102
17. López U., Trujillo L, Martinez Y, Legrand P, Naredo E, Silva S (2017) RANSAC-GP: Dealing with outliers in symbolic regression with genetic programming. Eur Conf Gene Program 10196:114–130
18. de Melo VV (2014) Kaizen programming. In: GECCO '14: Proceedings of the 2014 conference on Genetic and evolutionary computation, pp 895–902
19. Neri F, Eiben ÁE, Smith JE, Oca MAMD, Cotta C, Sudholt D (2012) Handbook of memetic algorithms
20. Nguyen QH, Ong YS, Krasnogor N (2007) A study on the design issues of memetic algorithm. In: IEEE Congress on evolutionary computation, CEC
21. Pawlak TP, Krawiec K (2018) Competent geometric semantic genetic programming for symbolic regression and boolean function synthesis. Evol Comput 26(2):1–36
22. 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, UK
23. Sabar NR, Aleti A (2017) An adaptive memetic algorithm for the architecture optimisation problem. In: Artificial life and computational intelligence: Third australasian conference, ACALCI 2017, geelong, VIC, Australia, January 31–February 2, 2017
24. Shao W, Pi D, Shao Z (2017) Memetic algorithm with node and edge histogram for no-idle flow shop scheduling problem to minimize the makespan criterion. Appl Soft Comput 54:164–182
25. Trujillo L, Z-Flores E, Juarez P, Legrand P, Silva S, Castelli M, Vanneschi L, Schütze O, Muñoz L (2018) Local search is underused in genetic programming. Gene Evol Computation 119–137
26. Vengatesan K, Bhaskar Ranjana M, Sanjeevikumar P, Mangrule R, Kala V, Pragadeeswaran (2018). Performance Analysis of Gene Expression Data Using Mann–Whitney U, Test ,701–709
27. Yang Y, Liu Y, Du L, Chang C, Wang D, Wang D (2010) MDE based memetic algorithm using openmp and its application in engineering project scheduling problems with dynamic due dates. In: International conference on logistics systems & intelligent management
28. Zflores E, Trujillo L, Schuetze O, Legrand P (2014) Evaluating the effects of local search in genetic programming. Springer International Publishing, Cham

29. Zhang C. Genetic programming for symbolic regression. Available at https://pdfs.semanticscholar.org/e5ee/ddd04b8344fd4f39a5836be686886c80df13.pdf
30. Zhang K, Cai Y, Fu S, Zhang H (2019) Multiobjective memetic algorithm based on adaptive local search chains for vehicle routing problem with time windows. Evol Intell 3:1–12
31. Zhong J, Liang F, Cai W, Ong YS (2018) Multifactorial genetic programming for symbolic regression problems. IEEE Trans Syst Man Cybern Syst PP(99):1–14

Jiayu Liang received her B.Sc. and M.Sc. degree in 2011 and 2014 respectively from University of Science and Technology Beijing, China. She got her Ph.D degree in Computer Science from Victoria University of Wellington, New Zealand. From 2018, she works as a lecturer in Tiangong University, China. Dr. Liang focuses on research areas, e.g. Evolutionary Computation, Multi-objective Optimization and Image Processing. She has published over 10 papers in top journals and international conferences in those areas.