Genet Program Evolvable Mach (2015) 16:133–150
DOI 10.1007/s10710-014-9230-4

Balanced Cartesian Genetic Programming via migration and opposition-based learning: application to symbolic regression

Samaneh Yazdani · Jamshid Shanbehzadeh

Received: 2 September 2013 / Revised: 9 July 2014 / Published online: 29 July 2014

Abstract The exploration–exploitation trade-off is an important aspect of evolutionary algorithms which determines the efficiency and accuracy of these algorithms. Cartesian Genetic Programming (CGP) is a generalization of the graph based genetic programming. It is implemented with mutation only and does not have any possibility to share information among solutions. The main goal of this paper is to present an effective method for balancing the exploration and exploitation of CGP referred to as Balanced Cartesian Genetic Programming (BCGP) by incorporating distinctive features from biogeography-based optimization (BBO) and opposition-based learning. To achieve this goal, we apply BBO's migration operator without considering any modifications in the representation of CGP. This operator has good exploitation ability and can be used to share information among individuals in CGP. In addition, in order to improve the exploration ability of CGP, a new mutation operator is integrated into CGP inspired from the concept of opposition-based learning. Experiments have been conducted on symbolic regression. The experimental results show that the proposed BCGP method outperforms the traditional CGP in terms of accuracy and the convergence speed.

Keywords Cartesian Genetic Programming · Biogeography-based optimization · Migration · Opposition-based learning · Exploration–exploitation trading-off

S. Yazdani (✉)
Department of Computer Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iran
e-mail: Samaneh.yazdani@gmail.com

J. Shanbehzadeh
Department of Computer Engineering, Faculty of Engineering, Kharazmi University, Tehran, Iran
e-mail: jamshid@khu.ac.ir

1 Introduction

Evolutionary algorithms have solved different problems, which are difficult to solve by using more traditional optimization algorithms. The research areas of evolutionary algorithms have been attracting researchers for many years. The performance of evolutionary algorithms depends on balancing between the exploration and exploitation which are obtained via randomized operations such as mutation and recombination. Mutation introduces new genetic material into an existing individual. So, mutation adds diversity to the genetic characteristics of population. Recombination shares information between two or more individuals [1, 2].

Genetic Programming (GP) is an evolutionary algorithm introduced by Koza [3, 4]. It has tree data structures as genotype. GP has been applied on various problems in different areas such as image processing and pattern recognition [5–9], biomedical science [10–13] to control engineering [14–16], robotics [17–19] and so on [20]. After introducing GP, researchers have been trying to develop new techniques to improve the performance of GP [21]. One such approach suggested recently by Miller and Thomson is Cartesian Genetic Programming (CGP) [22]. CGP uses directed graphs which are more general than the trees to represent programs. The CGP is implemented with mutation only [23], therefore, it may lack the exploitation ability. As the crossover techniques which were reported in [23] (i.e. like the simple point crossover that was applied in the CGP original integer representation) failed to improve the performance of CGP, Clegg et al. [23] introduced the new crossover. In order to apply this new operator, the CGP representation is modified in which a genotype is a fixed length list of real-valued numbers instead of the integer numbers. Each gene in genotype has a value in the interval [0, 1]. After applying the crossover inspired from the real-valued crossover operator in real-valued GAs, the decoding process from the real-valued genotype to integer based genotype should be performed. The results show that by applying a proper crossover, the performance of CGP can efficiently improve.

Biogeography-based optimization is a new evolutionary algorithm for global optimization introduced by Simon [24]. It is a population-based algorithm in which each solution of the population is a vector of integer. Since BBO has certain features in common with other biology-based optimization algorithms like genetic algorithms (GAs) and particle swarm optimization (PSO), BBO is applicable to the same type of problems which they claim to solve [24]. In [24] Simon compares the BBO performance on 14 benchmark functions with seven widely used population-based optimization methods. Results of his study show that BBO outperforms most of the other algorithms on most of the benchmarks. BBO applies the migration operator to share information among solutions. It has good exploitation ability due to the migration operator [25].

The new features of the proposed method which aims to balance the exploitation and exploration are: (1) Taking the migration operator of the BBO to share information among individuals. Applying this migration operator exempts us from modifying the CGP representation and dominant the lack of exploitation ability. (2) Driving a new mutation operator inspired by the concept of opposition-based learning. This new mutation operator enhances the exploration ability of CGP.

To illustrate the effect of the proposed method, another search method is used instead of the 1 + lambda search method applied with the traditional CGP. Experiments have been tested on some of the symbolic regression problems chosen from literatures. In addition, to fairly compare the proposed method with the traditional CGP three performance criteria have been utilized. These measures are success rate (SR), acceleration rate (AR), and the average of the best found solutions and the corresponding standard deviation as calculated after specific number of function evaluations has been reached. Furthermore, the influence of changing in both the number of nodes and population are investigated. The results demonstrate the importance of the exploration–exploitation trade-off.

The rest of this paper is organized as follows: Sect. 2 reviews the BBO algorithm. Section 3 introduces the traditional CGP and its problems. The proposed method called Balanced Cartesian Genetic Programming (BCGP) is also described in Sect. 3. Section 4 presents the experimental results and discussions. Finally, conclusions and directions for future investigations are given in Sect. 5.

2 Biogeography-based optimization

Biogeography-based optimization is a new population-based algorithm for global optimization which was originally developed by Simon [24]. Each individual (called island or habitat) represents a solution for the problem and is comprised of solution features. These solution's features are called suitability index variables (SIVs) which are the same as genes in GA [24, 25]. The goodness of each solution is characterized by a habitat suitability index (HSI). HSI is equivalent to fitness in GA. Two main operators of BBO for improving the population are the migration (which includes both immigration and emigration) and mutation [24, 25].

Migration is an operator for probabilistically sharing features among solutions. High-HSI solutions are the good ones and tend to share their features with low-HSI solutions by emigration. Low-HSI solutions are the poor ones and accept new features from high-HSI solutions by the immigration. In BBO, each individual has its own immigration rate λ and the emigration rate μ. These rates indicate the probability that a solution is selected as an immigration or emigration habitat. A good solution relatively has a high μ and low λ and vice versa for the poor solution. BBO has good exploitation ability due to these characteristic of the migration. For each solution in each generation, immigration and emigration rates are adaptively determined based on the fitness of the solution as follows [24, 25]:

$$\lambda_i = I\left(1 - \frac{k(i)}{n_{pop}}\right) \tag{1}$$

$$\mu_i = E\left(\frac{k(i)}{n_{pop}}\right) \tag{2}$$

where $k(i)$ represents the rank of ith individual in an ordered list sorted based on the fitness of the population from the worst fitness to the best one, and n_{pop} is the

number of solutions in the population. E and I are respectively the maximum possible rates of the emigration and immigration which are most of the time set to 1 or close to it. Figure 1 illustrates the above-mentioned explanations for a population sorted based on the fitness of individuals [24, 29]. According to the already mentioned definitions, migration can be expressed as [25]:

$$H_{i,SIV} \leftarrow H_{j,SIV} \tag{3}$$

where H_i is selected as the immigration solution with immigration rate λ_i, and H_j is the individual selected as the emigration solution with the emigration rate μ_j. Equation (3) means that a solution feature of solution H_i is replaced by a feature from solution H_j [25].

The mutation in BBO is utilized to modify solution's features, SIVs. According to the mutation rate, a selected SIV in the ith solution, H_i, is replaced by a randomly generated one. This process can be described as follows:

```
for i=1 to n_pop do
    Use λ_i and μ_i to compute the probability p_i
    for j=1 to D do
        Select SIV H_{i,j} based on mutation probability m_i
        if H_{i,j} is selected then
            Replace H_{i,j} with a randomly generated SIV
        end
    end
end
```

where D is the number of SIV in each habitat, $H_{i,j}$ is the jth SIV of the individual H_i, and m_i is calculated from equation (4).

$$m_i = m_{\max}\left(1 - \frac{p_i}{p_{\max}}\right) \tag{4}$$

where m_{max} is the user defined maximum mutation probability, $p_{max} = \arg\max p_i$, $i = 1,\ldots,n_{pop}$, and p_i is the solution probability [24, 25]. More details about mutation are discussed in [24].

3 Balanced Cartesian Genetic Programming

Miller and Thomson developed CGP [22]. Although CGP is from GP family, it represents a program as a directed graph unlike the standard tree based GP. The main reason is the more generality of a graph in comparison with a tree [22]. This directed graph is defined by a rectangular grid of nodes with n_r and n_c number of nodes in each row and column respectively. n_r and n_c are user defined parameters. Yu and Miller [27] showed that CGP has been more effective when number of rows was chosen to be one. Hence, we choose the number of rows to be one through this paper.

Genotype in CGP is a fixed length list of integers that encodes the function and the connections of each node in the directed graph [22, 23]. To explain the genotype

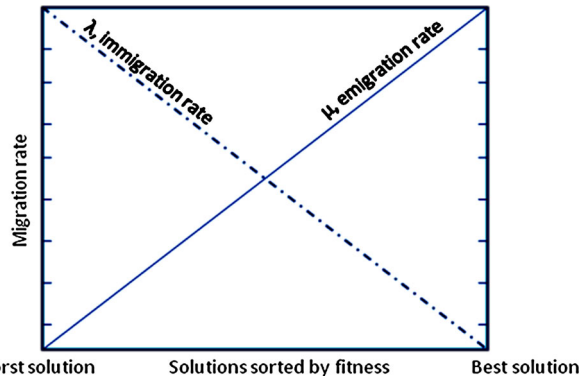

Fig. 1 Linear migration rates plotted versus the sorted population [29]

and corresponding phenotype of a program in CGP more clearly, the following example is considered. This example is borrowed from [23] which implements a function considered as $x^6 - 2x^4 + x^2$. As shown in Fig. 2, each node of the graph consists of genes. The first gene encodes the node function randomly selected from the function set. In this example, functions and corresponding indexes are {+(0), −(1), *(2), /(3)}. The remaining genes in the node encode where in the graph the node takes its inputs from. The nodes take their inputs from either output of a previous node or from the primitive program inputs (terminals). Each of inputs is labeled with the consecutive number from {0 (x), 1 (1)…}.

The number of previous columns of nodes which may have their output connected to a node in the current column is defined by levels-back parameter. Notice that the primary inputs are treated in the same way as node outputs [22]. In this example levels-back is equal to the maximum number of columns. Thus, nodes can connect to any previous nodes in left.

The program output is taken from the node output 8. Any encoded nodes in genotype can be either connected or disconnected. In Fig. 2, gray nodes are not connected to the program output and are inactive. Therefore, in contrary to the genotype, phenotypes (programs) in CGP have a variable length.

As mentioned above, CGP may lack the exploitation ability because it is implemented with mutation only. Some mutation operators include the point mutation, insert-node and delete-node [28]. Some attempts have been done to introduce an effective crossover for CGP when CGP representation is in its original form. However, all of them are failed to improve the performance of CGP. In [23], the crossover method which improved the performance of CGP was developed. However, to incorporate this type of crossover operator into CGP requires a modification to CGP representation itself [23].

Introducing a proper crossover technique can increase the exploitation of CGP and speed up its convergence considerably. The goal of this paper is to improve the performance of CGP by balancing its exploration and exploitation. To reach this purpose, we propose a new method which is called BCGP (Balanced Cartesian Genetic Programming) which is described in the following subsections.

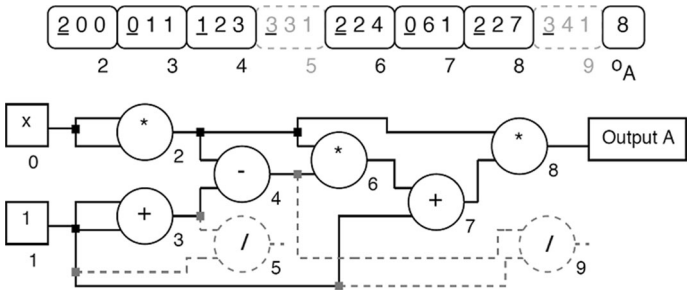

Fig. 2 Genotype and corresponding phenotype of a CGP for the function $x^6 - 2x^4 + x^2$. First gene in each node (*underlined gene*) encodes the function. The function set is {+=0, −=1, *=2, /=3}. The two inputs are 0(x) and 1(1) [23]

Algorithm 1 **B**alanced **C**artesian **G**enetic **P**rogramming (BCGP)

% λ_i and μ_i are immigration and emigration rates of i-th individual. They are calculated from equations (1) and (2) respectively.

Create the initial population P randomly
Evaluate the fitness for each individual in P

NOG=0 % number of generations
while ((Best_Fitness_Value_So_Far>VTR) and (NOG <= $M_{generation}$))
  Sort population from worst to best
  Compute immigration rate λ and emigration rate μ for each individual based on fitness value

  for each individual i=1…n_{pop} **do**
    for each gene k=1…D **do**
      Select individual P_i with probability $\propto \lambda_i$
      if P_i is selected **then**
        Use μ to probabilistically select the emigrating individual P_j
        $C_{i,k}$= $P_{j,k}$
      else
        $C_{i,k}$= $P_{i,k}$
      end
    end
  end

  Mutate the offspring with the mutation operator, which will be shown in Algorithm 2

  for i=1 to n_{pop} **do**
    Calculate the fitness value of $C_i(t)$
    $P_i(t+1)$= Fittest individual from {$P_i(t)$, $C_i(t)$}
  end

  NOG= NOG+1
end

3.1 Main procedure of BCGP

By employing the migration operator of BBO and being inspired by the concept of opposition-based learning as mutation, BCGP is developed to improve the performance of traditional CGP (see Algorithm 1). In BCGP, each individual is presented by a D dimensional integer vector. In Algorithm 1, VTR is value to reach, NOG is number of generations, and $M_{generation}$ is the maximum number of generations. n_{pop} is the size of the parent population P, $P_i,_k$ is the kth gene of the ith individual in the P, and C_i is the ith member of the offspring population C. The proposed method is an elitist method. Namely, if applying the proposed method yields a better offspring, parent is replaced with it. It is important to mention that the parent will be replaced with an offspring even if the offspring receives the same fitness value.

3.2 Migration

Based on the explanation mentioned in Sect. 2, migration has good exploitation ability, and it can utilize the population information effectively. In order to make a way for sharing information among solutions in CGP, the migration operator is integrated into CGP. Since CGP uses a vector of integers as a genotype just like BBO, we can use the migration operator without any modifications. Migration can be expressed in BCGP as:

$$P_{i,gen} \leftarrow P_{j,gen} \tag{5}$$

where P_i is the immigrating individual and P_j is the emigrating individual.

3.3 Mutation

In order to improve the exploration ability of the proposed method, the mutation operator which is composed of two types of mutation is integrated into BCGP. These types of mutation are different in terms of power of exploration and stochastic.

3.3.1 Type 1: Simple mutation

In type 1, the value of a gene is replaced by a randomly generated value from a valid integer interval.

3.3.2 Type 2: Opposition-based learning based mutation (OBL mutation)

Initial population in evolutionary algorithms is often created randomly. The computation time is related to the distance of these initial guesses from the

optimal solution. The chance of starting with a closer solution is increased by checking the opposition guesses at the same time. This is the main idea behind opposition-based learning proposed by Tizhoosh [26]. In opposition-based optimization, fitter one between guess or opposite guess can be chosen to create better initial population or better generation to accelerate evolutionary algorithms convergence. We use the concept of quasi-opposition-based learning as a mutation. As it has been shown in [30], quasi-opposite points have a better convergence rate than opposite points [29]. The value of the kth gene of the ith member of the offspring population C is updated as follows:

$$\begin{aligned} op_k &= Min_k^C + Max_k^C - C_{i,k} \\ M_k &= \frac{Min_k^C + Max_k^C}{2} \\ C_{i,k} &= \begin{cases} round(M_k + (op_k - M_k) \times rand(0,1)) & if \quad C_{i,k} < M_k \\ round(op_k + (M_k - op_k) \times rand(0,1)) & otherwise \end{cases} \end{aligned} \tag{6}$$

where Min_k^C is the minimum value of the kth gene in the current offspring population and Max_k^C is the maximum value of the kth gene in the current offspring population C. So, the maximum and minimum values of each variable (gene) in the current offspring population are used. As explained in [30], using this interval instead of predefined interval boundaries of variables helps to concentrate on the current reduced search space and use the obtained knowledge of converged population. M_k is the middle point, round(x) rounds x to the nearest integer, and rand(0,1) is a random number uniformly distributed between 0 and 1. It is important to notice that in order to employ OBL in integer coding, we add round to the relation of OBL.

The core idea of the proposed mutation type is that when the time increases, the search space is shrunken. OBL mutation operator is able to use the knowledge of the current reduced space. The mutation operator is described in Algorithm 2. m_r in Algorithm 2 is the mutation rate.

3.4 Differences between BCGP and CGP

There are three main differences between CGP and BCGP: (1) there are no ways for CGP to share information among individuals while BCGP using the migration operator can utilize the population information effectively. (2) In addition, the mutation operator in BCGP which is composed of two types of mutation tends to increase the diversity of population and improves the exploration ability. (3) Operators shown in Algorithm 1 can balance the exploration and the exploitation of BCGP while CGP may lack the exploitation ability.

Algorithm 2 Mutation

p_r=0.5;

for each individual in the offspring population i=1… n_{pop} **do**
  select randomly, m_r% of genes of C_i % every gene has an equal chance to be mutated
  for each gene k=1 . . . D
    if k-th gene has been selected
      if *rand* (0,1) <p_r
        $C_{i,k}$= randomly select number from {a,…,b} % Mutation Type 1. a,b are minimum and maximum valid value number for k-th gene
      else % Mutation Type 2

$$op_k = Min_k^C + Max_k^C - C_{i,k}$$

$$M_k = \frac{Min_k^C + Max_k^C}{2}$$

        if $C_{i,k} < M_k$
          $C_{i,k}$= $round(M_k + (op_k - M_k) \times rand(0,1))$
        else
          $C_{i,k}$= $round(op_k + (M_k - op_k) \times rand(0,1))$
        end
      end
    end
  end
end

4 Simulation result and analysis on symbolic regression problems

To evaluate the performance of BCGP with traditional CGP, five real-valued symbolic regression problems [3, 21–23, 31] which are shown in Table 1 are used. The symbolic regression involves finding a mathematical expression that relates the independent variables to the dependent variable for a given finite sampling interval. In other words, it involves finding the hidden relation among the variable/variables and the target concept.

In order to compare the methods, the following parameters have to be determined: terminal set, function set, mutation rate (m_r), maximum number of nodes, and maximum *NOG*s, $M_{generation}$. These parameters used in this study have been discussed in detail in [3, 21–23]. They are recommended as follows: terminal set of {1,X}, function set consisting of {+, −, *, /, sin, cos}, mutation rate of 3 %, maximum number of nodes of 50, and for the $M_{generation}$, we use 500 for our proposed method and 6,250 for traditional CGP. These parameters remain unchanged unless it is mentioned.

As can be seen, the maximum number of generations is set differently because CGP and BCGP employ different search methods. Since the search method over the CGP representation is very important and must be carefully chosen, we apply the (1 + lambda) evolutionary strategy[1] which most literature used it. Lambda is set to

[1] CGP work best using 1 + lambda.

Table 1 Symbolic regression functions

Functions	Fitcases
$F_1 = x^6 - 2\,x^4 + x^2$	50 random points $\subseteq [-1,1]$
$F_2 = x^5 - 2\,x^3 + x$	50 random points $\subseteq [-1,1]$
$F_3 = x^4 + x^3 + x^2 + x$	20 random points $\subseteq [-1,1]$
$F_4 = sin(x^2)\,cos(x) - 1$	20 random points $\subseteq [-1,1]$
$F_5 = sin(x) + sin(x + x^2)$	20 random points $\subseteq [-1,1]$

4 [32]. In order to equate the number of evaluations in each method, the $M_{generation}$ for CGP is set to 6,250.

Moreover, for all regression problems, experiments are repeated 100 times and average results are shown. Note that the fitness value of each individual is defined here as the sum of absolute errors between its values and the true function value over all fitness cases. Fitness cases are randomly selected from specified intervals which are shown in Table 1. The stopping condition is to find a value smaller than the value to reach (VTR) before reaching the maximum number of generations $M_{generation}$. We set VTR to 10^{-6}.

In order to have a fair comparison, three performance criteria are utilized. These criteria are:

- The Mean and standard deviation (St-d) of errors in 100 runs.
- The number of runs for which the algorithm successfully reaches the VTR for each test function is measured as the success rate (SR) [33].
- In order to compare convergence speeds, another measure called the acceleration rate (AR) is used [33]. As mentioned above, we must employ two different search methods in our proposed method and the traditional CGP. To have a fair comparison, we should compare BCGP and CGP in the same number of evaluations. We show AR as follows, based on the *NOG* and the number of individuals:

$$AR = \frac{NOG_{CGP} \times lambda}{NOG_{BCGP} \times n_{pop}} \tag{7}$$

where *NOG* is the number of generations. The number of generations multiplied by the population size is equal to the number of function calls (NFCs) which indicates the convergence speed. A smaller NFC means higher convergence speed. The average number of NFC over 100 runs is used for minimizing the effect of stochastic. $AR > 1$ means BCGP is faster.

4.1 Influence of migration on the performance of BCGP

In our first experiment, we investigate only the influence of applying migration on the performance of CGP. We call it BCGP1. In our proposed method, the size of the

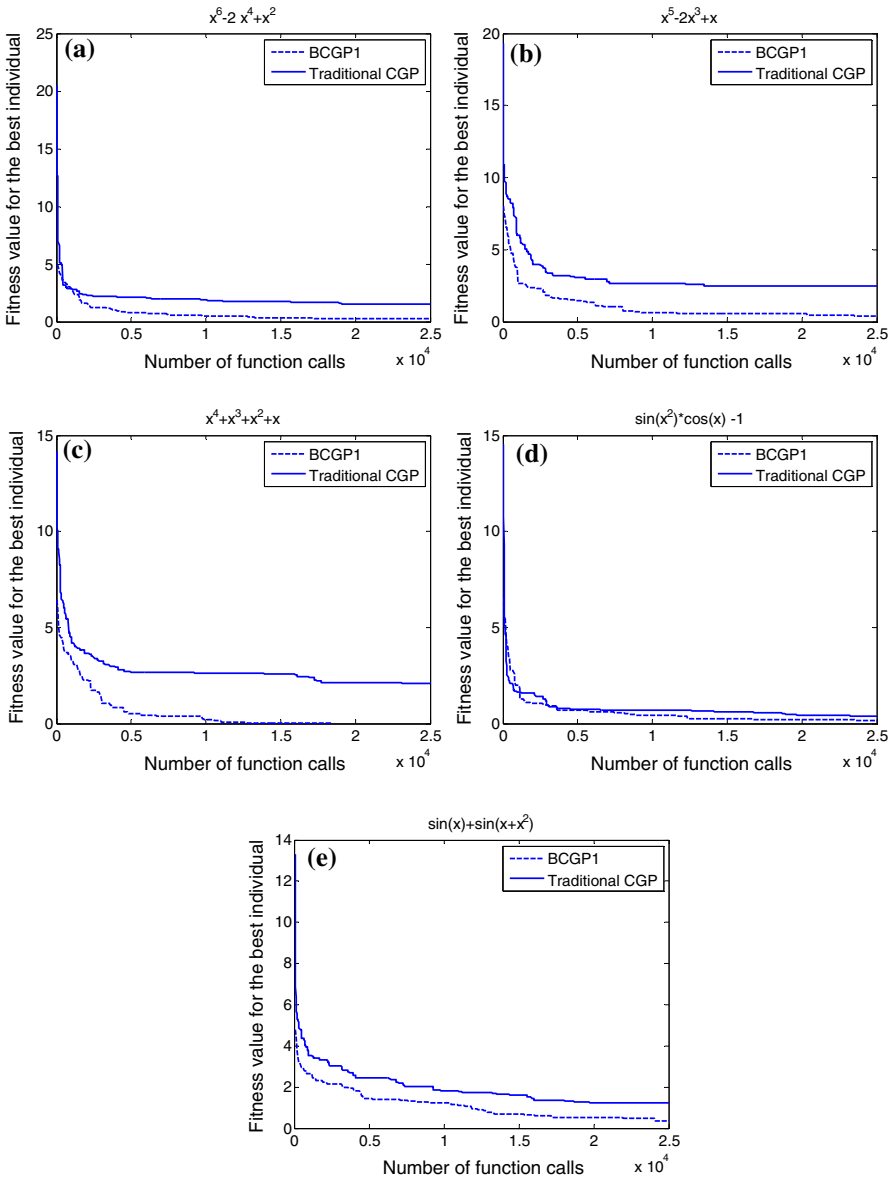

Fig. 3 Mean error curves of BCGP1 and traditional CGP for five regression problems. Results obtained from averaged over 100 independent runs

population is 50 and at each generation 50 offspring are created and as discussed in Algorithm 1, the better individual between *i*th parent and *i*th offspring survives for the next generation.

Figure 3 depicts average mean errors found by two algorithms over 100 runs for the benchmark functions. The average results of 100 independent runs of

Table 2 Comparison of BCGP1 and traditional CGP

	BCGP1			CGP			BCGP1 versus CGP
	Mean	St-d	SR	Mean	St-d	SR	AR
F_1	0.241	0.228	31	1.532	1.230	10	1.142
F_2	0.363	0.265	29	2.452	2.622	0	1.047
F_3	0	0	100	2.083	2.207	10	3.346
F_4	0.178	0.123	27	0.386	0.378	0	1.071
F_5	0.369	0.419	40	1.22	0.792	0	1.161

Table 3 Comparison of BCGP and CGP

	BCGP1			CGP			BCGP1 versus CGP
	Mean	St-d	SR	Mean	St-d	SR	AR
F_1	0.021	0.055	81	1.532	1.230	10	1.828
F_2	0.187	0.202	50	2.452	2.622	0	1.258
F_3	0	0	100	2.083	2.207	10	3.398
F_4	0.139	0.145	30	0.386	0.3789	0	1.24
F_5	0.211	0.290	50	1.22	0.792	0	1.330

BCGP1 and CGP on all benchmark functions are summarized in Table 2. As demonstrated in Fig. 3 and Table 2, applying the migration operator improves the performance of the traditional CGP in terms of convergence speed and accuracy significantly. In all figures, the horizontal axis represents the number of function calls in CGP (or BCGP1), and a vertical axis is the fitness value for the best individual which is averaged over 100 runs. We use number of function calls for comparison between algorithms because they have the same maximum number of function calls.

4.2 Comparison between CGP and proposed method (BCGP)

In this section, we try to prove the effectiveness of our proposed method. The population size for our proposed method is 50 as mentioned in Sect. 4.1. Table 3 summarizes the BCGP and CGP performances on the benchmark functions. It is obvious that BCGP performs significantly better than CGP consistently with respect to all three criteria for all regression problems.

Figure 4 depicts average convergence curves of CGP, BCGP1 and BCGP (over 100 runs) for all regression problems. According to Fig. 4, we can see that: first, the traditional CGP may trap into local optima while the proposed algorithm can locate a good near-global optimum (because the proposed operator allows CGP to escape from poor local optima). Second, we can see that BCGP is better than BCGP1. The comparison between, BCGP and BCGP1 on all regression according to Fig. 4 illustrates the effectiveness of the OBL mutation.

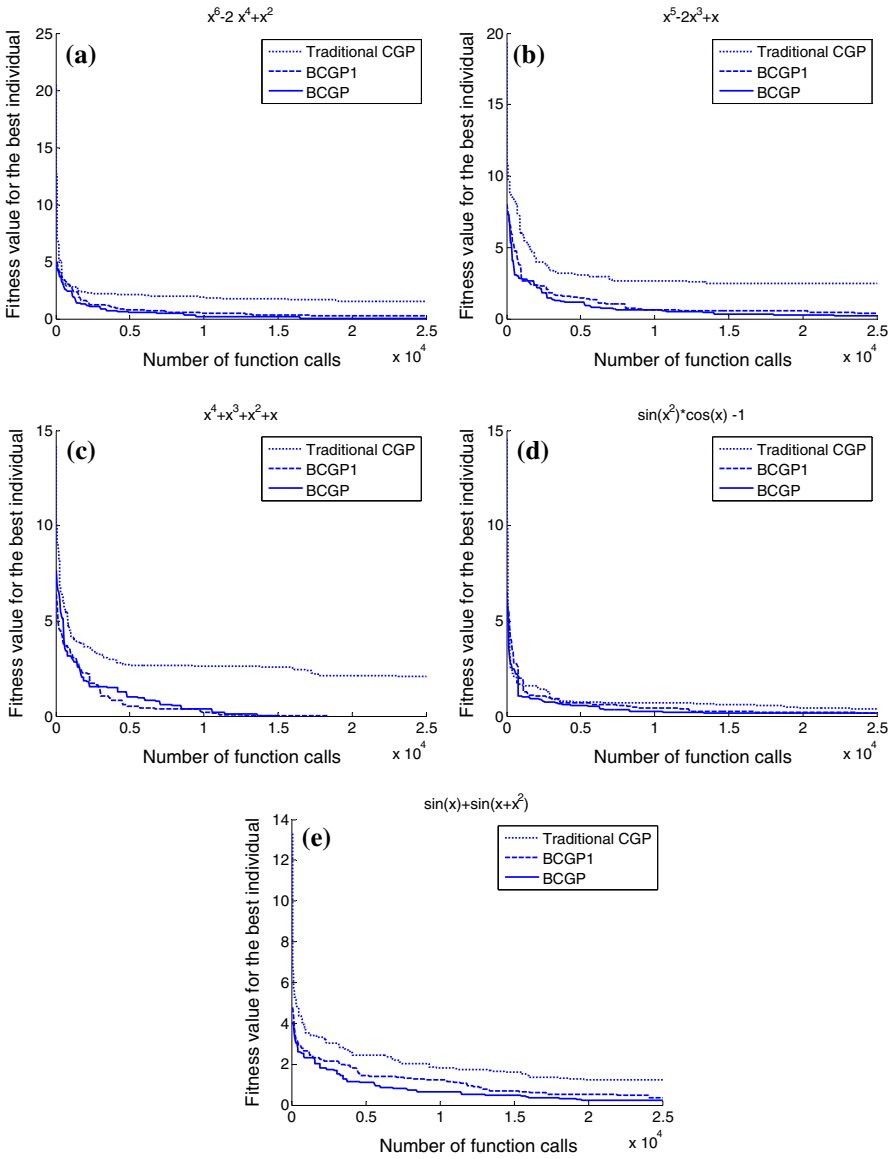

Fig. 4 Mean error curves of traditional CGP, BCGP1 and BCGP for five regression problems

Results indicate that the exploration–exploitation trade-off improves the performance of CGP. Third and in summary, BCGP converges faster than BCGP1 and CGP.

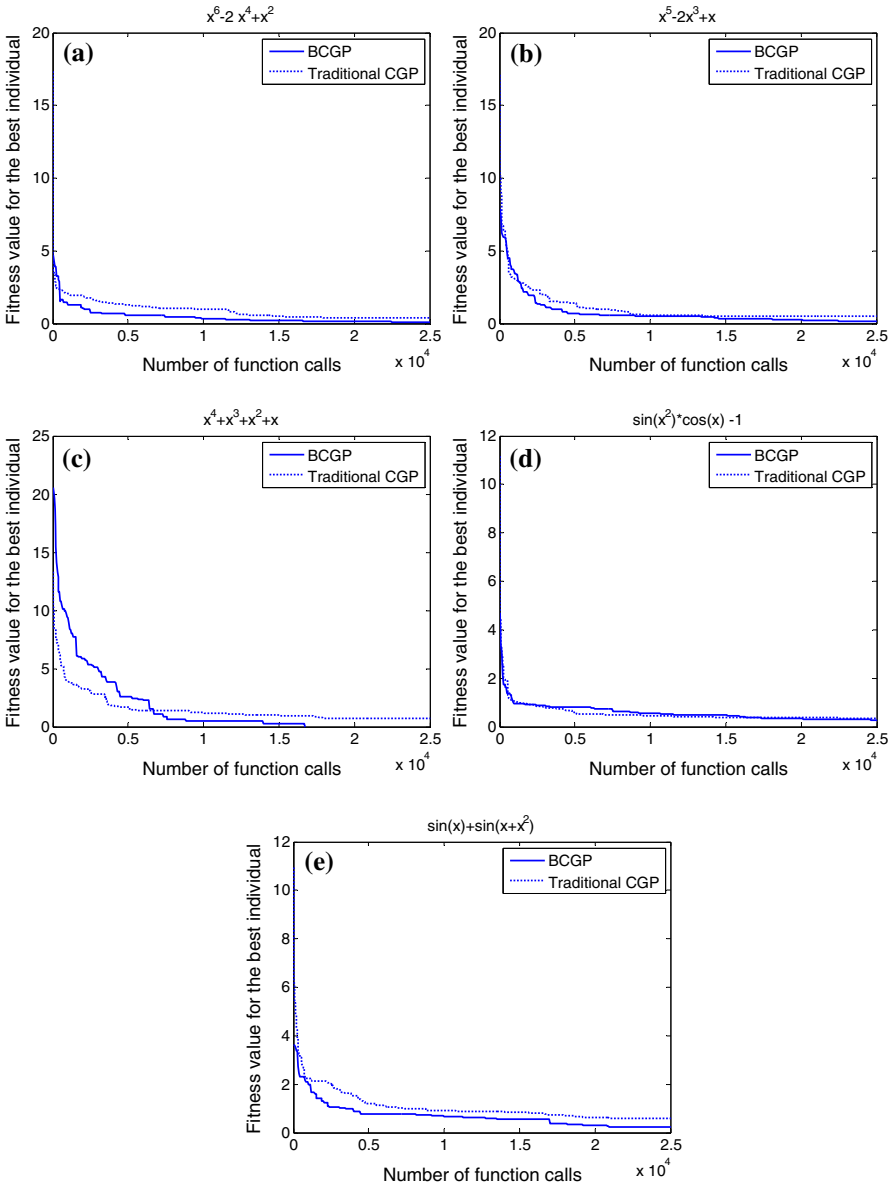

Fig. 5 Performance comparison between BCGP and traditional CGP with 100 nodes using the average error on the five regression problems

4.3 Effect of increasing the number of nodes

This section considers the influence of the number of nodes on the performance of two methods. In order to consider the effect of individual size on the performance of

Table 4 Comparison of BCGP and traditional CGP with 100 nodes

	BCGP1			CGP			BCGP1 versus CGP
	Mean	St-d	SR	Mean	St-d	SR	AR
F_1	0.079	0.107	40	0.369	0.200	0	1.644
F_2	0.167	0.178	50	0.481	0.464	0	1.474
F_3	0	0	100	0.709	1.07	10	3.563
F_4	0.253	0.100	0	0.325	0.255	0	1
F_5	0.217	0.273	50	0.596	0.857	10	1.185

Table 5 Influence of population size on the performance of BCGP

BCGP1	$n_{pop} = 150$			$n_{pop} = 50$		
	Mean	St-d	SR	Mean	St-d	SR
F_1	0.01	0.032	90	0.079	0.107	40
F_2	0.046	0.094	80	0.167	0.178	50
F_3	0	0	100	0	0	100
F_4	0.021	0.025	60	0.253	0.100	0
F_5	0	0	100	0.217	0.273	50

the two methods, similar to [23], the number of nodes is increased from 50 to 100 in all regression problems. Average convergence graphs are shown in Fig. 5. Results of solving the five regression problems are given in Table 4. Compared with CGP, BCGP performs better in terms of the quality of the final solutions and the convergence rate.

4.4 Influence of population size

The increase of individuals in population makes the initial diversity of population rise and allows large parts of the search space to be covered per iteration [1]. In order to consider the influence of population size on the performance of BCGP, the number of population is increased to 150 in BCGP. Other parameters are the same as in Sect. 4.3. The results for $n_{pop=}50$ and $n_{pop=}150$ are shown in Table 5. From Table 5, we can conclude that overall Mean criterion is decreased and SR is increased.

4.5 Comparison with the algorithm presented in [23]

In this section, the BCGP is compared with the real coded CGP presented in [23]. In Clegg et al. method, a new crossover technique was used. It was inspired by the real-valued crossover operator found in real-valued GAs. This method was tested on F_1 and F_2 (see Table 1). For two methods, the CGP basic parameters are as follows:

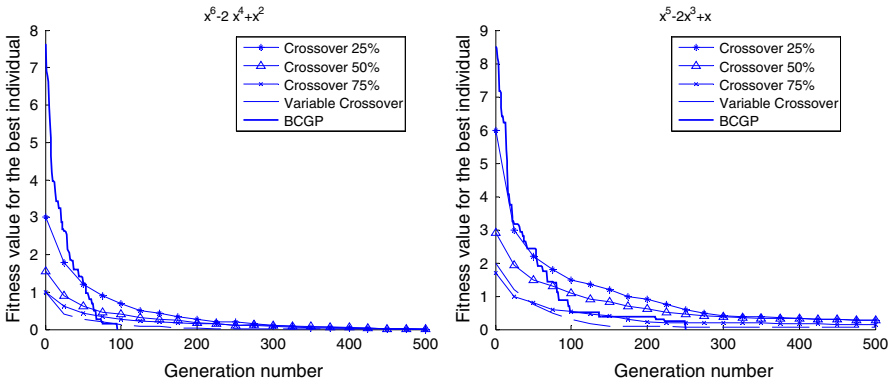

Fig. 6 Mean error curves of BCGP and Clegg et al. method considered with various crossover rates, for F_1 and F_2 symbolic regression problems

- Population size: $n_{pop} = 50$;
- Function set: {+,−,*,/};
- Terminal set: {1,X};
- Maximum number of nodes: 50;
- $M_{generation}$: 500;

Other parameters of BCGP are the same as mentioned in Sect. 4. The method presented in [23] was performed with different rates of crossover (such as 25 and 50 %) which are reported in Fig. 6. Since the maximum number of generations is the same as what used in two algorithms, they are compared on the basis of generations. According to Fig. 6, it is observed that the relative performance of BCGP is better than the method of Clegg et al. in terms of convergence speed and solution accuracy.

5 Conclusion

This paper highlights the importance of balancing the exploration and exploitation to improve the performance of CGP. CGP utilizes only the random mutation, therefore, it may lack the exploitation ability. In this paper, Balanced Cartesian Genetic Programming (BCGP) is proposed as a method to balance the exploration and exploitation ability of CGP. To dominate the lack of exploitation ability of CGP, the migration operator of BBO is integrated in BCGP. The main reasons for implementing the migration operator to share information among individuals in CGP are based on two considerations. First, this operator is easily applicable in CGP without any modification in CGP representation. Second, the migration operator has good exploitation ability, and it can utilize the population information effectively. Furthermore, a new mutation type namely opposition-based learning based mutation (OBL mutation) is applied in the mutation

operator of BCGP. The OBL mutation is defined on the basis of the concept of opposition-based learning to enhance the exploration ability of BCGP. The OBL mutation uses the knowledge of the population to reduce the exploration time as time increases.

The proposed method (i.e. BCGP) were tested on symbolic regression problems and compared with traditional CGP. In this study, three performance criteria have been utilized to fairly compare the algorithms. They are success rate (SR), acceleration rate (AR) and the average of the best found solutions and the corresponding standard deviation as calculated after specific number of function evaluations has been reached. AR is applicable to compare the convergence speeds of two methods when one generation of each algorithm executes the different number of evaluations. Experimental results show that by using OBL mutation and migration operators, BCGP outperforms CGP in terms of accuracy and the convergence speed.

Experiments that we performed verify that our proposed operators improve the CGP performance. For all cases, BCGP performs better than CGP with respect to the performance measures such as SR. Also, the effect of increasing the length of the individual and the population size on the performance of BCGP is investigated.

Our future will consist of applying proposed method on the real-valued representation of CGP which was presented in [23]. Future work can be investigated about how other types of migration are incorporated into CGP to improve its performance. Another direction for future work will be to extend the method so that it can be applicable as the preprocessing method to discover the relation among features by constructing new features in order to facilitate the learning of the target concept in the classification task.

References

1. A.P. Engelbrecht, *Computational Intelligence, an Introduction*, 2nd edn. (Wiley, New York, 2007)
2. T. Weise, *Global Optimization Algorithms—Theory and Application*. Available: http://www.it-weise.de, 2009
3. J.R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (MIT Press, Cambridge, 1992)
4. J.R. Koza, *Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems*. Technical Report STAN-CS-90-1314, Department of Computer Science, Stanford University, 1990
5. J.M. Daida, T.F. Bersano-Begey, S.J. Ross, J.F. Vesecky, Evolving feature-extraction algorithms: adapting genetic programming for image analysis in geoscience and remote sensing. Geosci. Remote Sens. Symp. Remote Sens. Sustain. Future **4**, 2077–2079 (1996)
6. D. Howard, S.C. Roberts, *Object Detection by Multiple Textural Analyzers*. Evolutionary Computation, Washington, DC, vol. 2, pp. 850–854, 6–9 July 1999
7. M. Kotani, M. Nakai and K. Akazawa, Feature extraction using evolutionary computation. In *Evolutionary Computation*, Washington, DC, USA, vol. 2, pp. 1230–1236, 6–9 July 1999
8. J. Koza, Simultaneous discovery of detectors and a way of using the detectors via genetic programming. in *IEEE International Conference*, vol. 3, pp. 1794–1801, 28 March–1 April, 1993
9. M.M. Rizki, M.A. Zmuda, L.A. Tamburino, Evolving pattern recognition systems. IEEE Trans. Evol. Comput. **6**(6), 594–609 (2002)

10. F. Fernandez, M. Tomassini, L. Vanneschni, Saving computational effort in genetic programming by means of plagues. in *Proceedings of the 2003 Congress on Evolutionary Computation*, vol. 3, pp. 2042–2049, 8–12 Dec 2003
11. H. Guo, A.K. Nandi, Breast cancer diagnosis using genetic programming generated feature. in *2005 IEEE Workshop on Machine Learning for Signal Processing*, pp. 215–220, 28–30 Sept 2005
12. J.H. Hong, S.B. Cho, The classification of cancer based on DNA microarray data that uses diverse ensemble genetic programming. Artif. Intell. Med. **36**(1), 43–58 (2006)
13. R. Seehuus, Protein motif discovery with linear genetic programming. Knowledge-based Intelligent Information and Engineering Systems, PT 3. in *Proceedings Lecture Notes in Artificial Intelligence*, vol. 3683, pp. 770–776, 2005
14. J. Imae, S. Nakatani, J. Takahashi, A design method for optimal controllers of minimax problems: a genetic programming approach. in *American Control Conference*, vol. 6, pp. 5394–5399, 4–6 June 2003
15. Y. Jessen, M.A. Keane, J.R. Koza, Automatic design of both topology and tuning of a common parameterized controller for two families of plants using genetic programming. in *Proceedings of Eleventh IEEE International Symposium on Computer-Aided Control System Design (CACSD) Conference and Ninth IEEE International Conference on Control Applications (CCA) Conference*, vol. 11, pp. 234–242, 25–27 Sept 2000
16. K.A. Marko, R.J. Hampo, Application of genetic programming to control of vehicle systems. in *Intelligent Vehicles Symposium*, Detroit, MI, USA, vol. 1, pp. 191–195, 29 June–1 July 1992
17. K.-J. Lee, B.-T. Zhang, *Learning Robot Behaviors by Evolving Genetic Programs. Industrial Electronics Society*. Control and Instrumentation (IECON-2000), vol. 4, pp. 2867–2872, 2000
18. M.C. Martin, Genetic programming for real world robot vision. in *Intelligent Robots and System, IEEE International Conference*, vol. 1, pp. 67–72, 30 Sept–5 Oct 2002
19. C.H. Messom, M.G. Walker, Evolving cooperative robotic behaviour using distributed genetic programming. Control Autom. Robot. Vision **1**, 215–219 (2002)
20. P. Kouchakpour, A. Zaknich, T. Braunl, Population variation in genetic programming. Inf. Sci. **177**(17), 3438–3452 (2007)
21. J.A. Walker, J.F. Miller, Automatic acquisition, evolution and reuse of modules in Cartesian genetic programming. IEEE Trans. Evol. Comput. **12**(4), 397–417 (2008)
22. J. F. Miller, P. Thomson, Cartesian genetic programming. in *Proceedings of 3rd European Conference on Genetic Programming (EuroGP 2000)*, vol. 1802, Lecture Notes in Computer Science, pp. 121–132, Edinburgh, 2000
23. J. Clegg, J.A. Walker, J.F. Miller, A new crossover technique for Cartesian genetic programming. in *Proceedings of GECCO*, pp. 1580–1587, 2007
24. D. Simon, Biogeography-based optimization. IEEE Trans. Evol. Comput. **12**, 702–713 (2008)
25. H. Ma, D. Simon, "Blended biogeography-based optimization for constrained optimization. Eng. Appl. AI **24**(3), 517–525 (2011)
26. H. Tizhoosh, Opposition-based learning: a new scheme for machine intelligence. in *Proceedings of International Conference on Computational Intelligence for Modeling Control and Automation*, vol. 1, pp. 695–701, 2005
27. T. Yu, J.F. Miller, Neutrality and the evolvability of boolean function landscape. in *Proceedings on EuroGP*, pp. 204–217, 2001
28. J. F. Miller, What bloat? Cartesian Genetic Programming on Boolean problems. in *Genetic and Evolutionary Computation Conference Late Breaking Papers*, pp. 295–302, 2001
29. M. Ergezer, D. Simon, D. Du, Oppositional biogeography-based optimization. in *Proceedings on SMC*, pp. 1009–1014, 2009
30. S. Rahnamayan, H.R. Tizhoosh, M.M.A. Salama, Quasi-oppositional differential evolution. in *Proceedings of IEEE Congress on Evolutionary Computation*, pp. 2229–2236, 2007
31. N.Q. Uy, N.X. Hoai, M. O'Neill, R.I. McKay, E. Galvan- Lopez, Semantically-based crossover in genetic programming: application to real-valued symbolic regression. Genet. Program Evolvable Mach. **12**(2), 91–119 (2011)
32. J. F. Miller, *Cartesian Genetic Programming*. (Springer, Berlin, 2011)
33. S. Rahnamayan, H.R. Tizhoosh, M.M.A. Salama, Opposition-based differential evolution. in *IEEE Transactions on Evolutionary Computation*, vol. 12, pp. 64–79, 2008