Neurocomputing 137 (2014) 293–301

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

An improved Gene Expression Programming approach for symbolic regression problems

YuZhong Peng, ChangAn Yuan*, Xiao Qin, JiangTao Huang, YaBing Shi

Key Lab of Scientific Computing & Intelligent Information Processing in Universities of Guangxi, Guangxi Teachers Education University, Nanning 530001, China

ARTICLE INFO

Article history:
Received 7 February 2013
Received in revised form
23 April 2013
Accepted 29 May 2013
Available online 18 February 2014

Keywords:
Genetic computing
Gene Expression Programming
Evolutionary algorithm
Symbolic regression
Data modeling

ABSTRACT

Gene Expression Programming (GEP) is a powerful evolutionary method for knowledge discovery and model learning. Based on the basic GEP algorithm, this paper proposes an improved algorithm named S_GEP, which is especially suitable for dealing with symbolic regression problems. The major advantages for this S_GEP method include: (1) A new method for evaluating individual without expression tree; (2) a corresponding expression tree construction schema for the new evaluating individual method if required by some special complex problems; and (3) a new approach for manipulating numeric constants so as to improve the convergence. A thorough comparative study between our proposed S_GEP method with the primitive GEP, as well as other methods are included in this paper. The comparative results show that the proposed S_GEP method can significantly improve the GEP performance. Several well-studied benchmark test cases and real-world test cases demonstrate the efficiency and capability of our proposed S_GEP for symbolic regression problems.

1. Introduction

Gene Expression Programming (GEP) is developed by a Portuguese scientist, named Ferreira, in 2001, which were derived and improved from Genetic Algorithm (GA) and Genetic Programming (GP) [1]. It is a new revolutionary member of the genetic computing family, benefiting from the genetic expression of the knowledge discovery technologies, owning to the merits of GP and GA, that evolves computer programs. In fact, they can take many forms such as mathematical expressions, neural networks, decision trees, polynomial constructs, logical expressions, and so on [1].

With simple, linear and compact chromosomes and easy genetic operators, GEP is a powerful global search tool. Since Ferreira released the first GEP research results, GEP has become an active research area of evolutionary computation and has been applied in many fields and well solved a large variety of complex problems, such as classification [2,3], symbolic regression and function mining [4–6], time-series analysis [7], optimization [8], and so on.

In recent years, numerous researchers have investigated GEP and proposed a series of improved GEP methods, processing data in specific fields with high effectiveness and efficiency. Li et al. [9], proposed a prefix K-expression structure to try to preserve good structures, which achieves a better convergence and efficiency in the classification; Duan et al. [10], posed a new dynamic adjustment of individual coding length through the ORF filter operator to reduce the situation of GEP lowering efficiency caused by an overly long string of individuals. With all the theories aforementioned, however, the primitive GEP should be applied to describe the genotype into the expression tree, and further to traverse the expression tree to calculate the fitness. As the tree construction and traverse were of very time-consuming operation, it greatly affected the efficiency of the algorithm; Elena et al. [11], presented an adaptive GEP algorithm, which automatically adapted the number of genes used by the chromosome. The adaptation process taken place at chromosome level, allowing chromosomes in the population to evolve with different number of genes to reduce the computational effort; Ryana and Hiblerb [12] presented a Robust Gene Expression Programming which used the simple grammar of prefix expressions and the simple encoding of bit vectors, reaping the benefits of encoding the expressive structures of trees and the power of breaking the “phenotype barrier”.

Symbolic regression, namely symbolic function identification, is a function discovery approach for analysis and modeling of numeric multivariate data sets for a purpose of getting insights about data-generating systems [14]. Symbolic regression has had both successful academic [15–18]and industrial applications [19,20].

Based on the basic GEP proposed by Ferreira, this paper describes the power S_GEP, which is specifically suitable for

* Corresponding author. Tel.: +86 077 1561 2556.
E-mail address: yca@gxtc.edu.cn (C. Yuan).

http://dx.doi.org/10.1016/j.neucom.2013.05.062
0925-2312

symbolic regression problems. The S_GEP has several other improvements containing: a new method for decoding and evaluating chromosome based on our previous research in literature [13], and a corresponding expression tree (ET) constructing and its traversing schema, a new approach for manipulating constants. The proposed new method for decoding and evaluating chromosome does not require constructing and traversing the ET but directly using stacks to decode chromosome and evaluate the fitness, to reduce the time–space complexity. The proposed new approach for manipulating numeric constants improves the convergence of population. Experimental results obviously indicate that S_GEP outperforms classical GEP with less computational effort and higher effectiveness.

2. Preliminary material

2.1. Brief overview of symbolic regression

In detail, the task of regression is to identify the variables (inputs) in the data that are related to the changes in the important control variables (outputs), to express these relationships in mathematical models, and to analyze the quality and generality of the constructed models [14]. Symbolic regression differs from traditional regression since it does not rely on a specific a priori determined model structure. The only assumption made in symbolic regression is that the response surface can be described by an algebraic expression. Instead of the traditional approach where the model structure is fixed and the remaining free parameters are optimized, symbolic regression reformulates the regression problem as a search problem for the optimal model structure. Once a model structure of sufficient quality is found, traditional techniques can be used to find the optimal coefficients [15].

2.2. Brief overview of GEP

As the case with GP and GA, when using GEP to solve a problem, the main algorithm description is similar with GP and GA. And generally the five components: function set, terminal set, fitness function, GEP control parameters, and stop condition, all need to be specified.

Chromosomes of GEP are composed of one or several genes by connecting with operators. The gene is made up of a head and a tail with the head encompassing functions and terminals, and the tail containing only terminals, of which function set is formed by all the function symbol needed while solving problems. The terminals set is made up of the known symbols, variable or constant describing the problems. And the head length h, tail length t and the largest number of function arguments n must meet the following relations:

$$t = h \times (n-1)+1 \tag{1}$$

where, h is determined by the users according to the problems to be solved. The relationship ensures that no matter how the head is composed, genes can be decoded with the effective semantic meaning and valid expression. First, GEP codes the individuals into the linear string of fixed length. It encodes the operands to form the genome while doing optimization. Its coding rules can be simply described as the following: the algorithm builds the expression tree (ET) with genome according to their semantic, and traverses the ET from top to bottom, then from left to right, and the derived symbol serials are effective parts of the gene code (known as K-expression). This process is called decoding chromosomes. Some tail nodes may not be seen in the expression tree. Such redundant nodes accommodate the future operations of the genetic changes in the structure which may have left space in terms of the expression imposed on the string of genetic operations; we will resolve the semantic effective expression tree. All the GEP individuals are effective candidates of solution [1]. After the results of each generation evolution undergo the moderate evaluation of fitness function, high fitness individuals are retained, and have a higher chance to breed future generations by moving in cycles. The algorithm will not stop until a satisfactory solution or expected evolution algebra have been achieved.

The group in the evolutionary process needs to evaluate the individual with fitness. Generally, the traditional method of fitness calculation needs to transfer the representative individuals of chromosomes or genomes into the expression tree, and then traverse the expression tree and derive the corresponding mathematical expression to compute the value of the K-expression, finally, evaluate the fitness according to the fitness function. In each generation evolution, high fitness individuals are preserved with more chances to breed future generations, and repeatedly in this method. The algorithm will not stop until a satisfactory solution or expected evolution generations have been derived.

2.3. Constants

It is assumed that the creation of floating-point constants is necessary to do symbolic regression in general [16,18]. Ferreira introduced two approaches for symbolic regression in the original GEP [17]. The first algorithm does not include any constants in the terminal set, but relies on the spontaneous emergence of necessary constants through the evolutionary process of GEP. Whereas the second algorithm involves the ability to explicitly manipulate random constants by adding a random constant domain Dc at the end of the gene. Experiments have shown that the first algorithm is more efficient in terms of both accuracy of the evolved models and computational time for solving problems.

Because in many experiments, the explicit use of random constants results in considerably worse performance. Ferreira gives advice that GEP find or compose the most suitable combination of constants itself [17]. And some research presents some GEP-RNC variant method to fine-tuned constants [4,17,18,21]. But using them, many extra computational resources are required.

2.4. Traditional method of decoding the GEP

In GEP, the group in the evolutionary process needs to calculate the individual fitness to evaluate. But the traditional method of calculate the individual fitness value needs to transform the genotype of chromosome into expression tree, and then traverses expression tree and attains the corresponding mathematical calculation. Finally, we need to compute the fitness according to the fitness function. According to the basic principles of GEP, the usual practice is to read from left to right one by one of the corresponding genes in the chromosome of the characters with the first character as the tree root, and the remaining follow-up node structure of the tree as a branch added to the parent node. The level of the corresponding ET will be built with the top-down, from left to right. Then according to their semantic level traverse the nodes in ET, we will gain the valid string of gene encoding, then calculate the value of K-expression in the first order traverse of ET (this is the value of the gene, if this chromosome is one with a single gene, the value of the chromosome is the value of the gene; while the chromosome is the one with couple genes, the value of the chromosome gene is the value of each gene calculated according to certain rules), and finally in accordance with the established fitness function to evaluate fitness value of the chromosome. For example, the following is a GEP chromosome

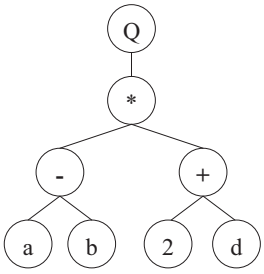

Fig. 1. Example of GEP Expression Tree (ET).

of length eight:

$$Q^{*}--+ab2dc \tag{2}$$

where Q denotes the square-root function; 2 is a numeric constant. According to the principles of GEP mentioned above, the expression tree of the chromosome described in Eq. (2) is shown in Fig. 1, and can be interpreted in a mathematical form as Eq. (3):

$$\sqrt{(a-b)^{*}(2+d)} \tag{3}$$

Then the value of the corresponding chromosome can be carried out by computing the mathematical expression.

2.5. Realization of conventional algorithm

The tree structure is usually used to realize the conventional algorithm for calculating the expression. Such algorithm takes the first character of K-expression for the root, and then in line with certain traversing sequence, the follow-up nodes of the K-expression construct the subtrees, and the subtree as a sub-branch added to their parents. And the algorithm recursively carries out the entire process until the expression completes constructing, finally, traverses and evaluates the constructed tree [1]. As Algorithm 1 shows below, it is a main framework for decoding algorithm of a gene from a chromosome.

The advantages of this algorithm in terms of calculating expression are that the tree structure is direct and clear, easy to understand. Shortcomings include: (1) establishing nodes need to assign much dynamic memory space, constructing expression tree needs to carry out frequent heap operation, the efficiency of the program significantly declines due to a large number of heap operation; and (2) tree structure is non-linear and the efficiency of computer systems addressing non-linear memory is not high.

Apparently, GEP converts the genome into ET, then traverse ET to evaluate the individual. This is the tree structure evaluation. Such evaluation method is time-consuming, in the mean time it takes up a lot of dynamic memory to store nodes of the tree because of constructing and traversing the tree. Thus, it greatly affects the efficiency and performance of the algorithm. This paper poses an improved GEP algorithm which can effectively alleviate drawbacks of method of decoding and evaluating and further improve the efficiency of GEP algorithm.

Algorithm 1. (conventional tree structural GEP algorithm):

```
Input: string of gene/chromosome
Output: evaluating value of gene/chromosome
Calculate-with-tree (string & chromstr) {
    buildET (Tree & T, chromstr);
    Return Cal-ET (Tree & T);
// traverse expression tree for calculating K expression
    }
Cal-ET (Tree & T) {
If (isoperator (root-> element))
      Return Cal-ET (root-> lchild) Op Cal-ET (root-> rchild);
      // individually traversing the various sub-tree, evaluating
      each sub tree
Else
      Return Value (root-> element);
   }
```

3. S_GEP algorithm

3.1. Numeric constants in S_GEP

In traditional GEP algorithms, the translation of the head/tail domain is done in the usual fashion, but, after translation, additional processing is needed in order to replace the question marks in the tree by the numeric constants they represent. S_GEP introduces a rather simple yet highly effective strategy to handle this problem, thus enabling GEP to discover functions with constant terms and accelerating convergence procedure. The strategies allow the constant to be a special terminal named as constant terminal, hereafter, detailed as follows:

Definition 1. (**Co-occurrence rate, COR**) in S_GEP, assumed the number of constant terminals in the same gene is n and the gene head length is h, then in this gene COR$=n/h$. Big COR would damage the gene expression ability.

(1) Some small prime and its reduced-scale with 10, as well as some random floating point constant, like,1, 2, 3, 0.1, 0.2, etc., are allowed containing in the terminals set. But it needs to modestly control the co-occurrence rate of these special terminals in a gene. So that these constant terminals can be useful for allowing solutions to be fine-tuned, and obtain optimum quickly.
(2) Some pre-defined constants, like, e or other user-defined values are allowed to be as special terminals in accordance with the problem to be solved. Also, it needs to modestly control the co-occurrence of the same special terminal in a gene too. So that these special terminals can be useful for allowing solutions to be fine-tuned in particular field problems, and obtain optimum quickly.
(3) Apparently, S_GEP only uses a node of ET and one mutation to form this pattern when some special pre-defined constants are allowed to be terminal. Surely, this method not only saves gene-bits but also reduces the genetic operation and arithmetic operations. Yet in conventional GEP, it would use many nodes of ET and many genetic operations are randomly combined to generate these pre-defined constants. So it can improve the algorithm efficiency of S_GEP, especially in particular problems.

3.2. Effective gene length

Ferreira borrowed the gene Open Reading Frame (ORF) in biology, which described the effective part of a biological gene to describe the GEP gene coding region [1]. It means that each GEP gene consists of two parts: the coding region and non-coding region, and not all elements of the gene appear in the corresponding expression tree. The coding region is the effective part of genes in the current generation. The start position of the ORF is the first symbol of the GEP gene, though the terminational position of the ORF usually is not the final symbol of the GEP gene. Therefore, the length of effective part of the gene is the length of the ORF, which is less than the gene's length.

Definition 2. (Effective gene length, *eL*). The effective length gene is the number of tree nodes in ET with the gene transforming from the genotype into a phenotype. Actually, it is the length of effective part of the gene. For example, the *eL* of the gene "Q*−+abcde" is 8, the "e" which of the chromosome segment of the tail is not used. The effective gene length of the gene "+−*Q/aaaaba" is 10. The *eL* is unambiguous and changeable in the evolutionary process, however the gene's length is fixed. The *eL* can be calculated through the following Algorithm 2 presented in this study.

So, when decoding and evaluating the chromosome, only computing for effective part of the gene can eliminate the waste of system resources causing by the computing of the non-coding region of the gene, and improve the efficiency of the algorithm.

Algorithm 2. (Effective gene length calculating algorithm)
Input: String of gene symbol
Output: *eL*
Step1: Initializing for parameters setting, Len=1, count=0;
Step2: Scanning the left first symbol of the gene;
Step3: Len=Len+1; If the current gene symbol is function then count=count −1+the number of arguments of current function, or else count=count −1;
Step4: if count<0,then go to the Step5, or else scanning the next symbol of the gene to right, and then go to the Step3;
Step5: *eL*=Len.
Programming end.

3.3. K-expression evaluation based on stack

The gene decoding expression of GEP is Polish notation. Therefore, it can be used as a stack directly for calculating the value of such operation expression, without first building a tree then traversing the tree for the evaluation of expression. The calculation of Polish Notation is much more concise in the algorithm than that of midfix expression. According to characteristics of the Polish Notation for value, this paper proposes the S_GEP algorithm. Its basic idea is as follows: finding the effective part of the gene, scanning the gene coding region of GEP from right to left with a stack, and then pushing corresponding data of terminals scanned into the stack. When each function has been scanned, the corresponding number of argument operand of top-stack should be popped from stack and the operation should be carried out. Then the computing results are pushed into the stack, scanning repeatedly until the final operational character of the expression has been disposed of. The value on top of stack is that of Polish Notation, that is the K-expression value. For example, as Eq. (4) (The 'Q' denotes a square-root computing; The '? 'denotes operator of condition, of which the largest number of operand is 3.) shows the evaluation of chromosome K-expression for the process as shown in Fig. 2 (What is circled by a red virtual circle is the evaluation of K-expression.).

$$Q?+abcdefg \tag{4}$$

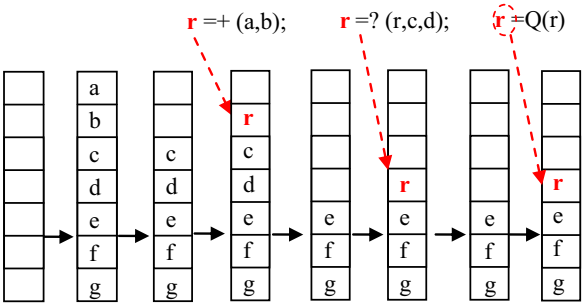

Fig. 2. Process of decoding and evaluating with S-GEP.

As Algorithm 3 shows below, it is main framework of S_GEP for decoding algorithm of a gene from a chromosome.

Is this stack-based method is valid for any chromosome of GEP? We prove it through Theorem 1 and its proof.

Algorithm 3. (S_GEP algorithm):

```
Calculate-with-SGDE (string & GeneSerial)
{
  Calculate the eL with Algorithm 2
  For (j=eL; j >=0; j–)
  {
    If (GeneSerial[j] is Terminals)
      mystack.push (value (GeneSerial[j]));
    Else if (GeneSerial[j] is Functions)
      mystack.push(operate(mystack.pull, mystack.pull,... ,
      mystack.pull));
  }
  Return mystack.pull;
}
```

Theorem 1. *In S_GEP algorithm, for the valid grammar of the GEP chromosomes, any time the number of operations is greater than, or equal to the corresponding number of operand of the function currently scanned.*

Proof. According to the valid grammar of GEP chromosomes with tail length t, head length h and the largest number of arguments the nature of the relationship is $t=h\times(n-1)+1=nh-h+1$. In the hypothetical extreme cases, the number of arguments of every function in the chromosome is the largest number of arguments in the function set, that is, every function has n argument, and the head is function at all, totaled h function. Then, in accordance with the basic idea of S_GEP can be shown as follows:

When scanning the first function and no stack operating, the number of arguments in the stack is: $nh-h+1$; When scanning the first function and having stack operating, the number of arguments in the stack is: $nh-h+1-n+1$; When scanning the second function and carrying out stack operating, the number of arguments in the stack is: $nh-h+1-2n+2$;

Similarly, when scanning to the function of No.h (that is the left-most function character of chromosomes) and just a stack operation, the number of operands in the stack is: $nh-h+1-nh+h=1$. So, after finishing scanning each function character and carrying out the corresponding stack operating, an operand remained in the stack, indicates that Theorem 1 is established. Similarly evidence, the number of arguments of every function in the chromosome is equal to or smaller than the largest number of arguments of the function set. After the function characters are all scanned and carried out corresponding stack operation, there is also greater than or equal number of operand to those in the stack. Therefore, Theorem 1 is established. □

Theorem 1 ensures to get the evaluation results of K-expression from the top of the stack after finishing scanning every valid chromosome. So the S_GEP algorithm can decode and evaluate every GEP chromosome of any valid grammar.

3.4. Analysis for algorithm complexity

The following is to take the K-expression computing of the single gene chromosome as an example to discuss the complexity of algorithms of traditional tree structure and of the S_GEP. In Algorithm 1, the running time is mainly spent on the recursive functional evaluation of the K-Expression resolved by traversing the ET. Calling for the Cal-ET function each time would require two recursions. The recursive formula equation of Algorithm 1 can be presented by Eq. (5) as follows:

$$f(n)=\begin{cases}1, & n=1\\ 2f(n/2)+cn, & n\geq 2\end{cases} \tag{5}$$

where, the n is constant and the length of the chromosome as well.

Recursive formula used to evaluate Eq. (5), gaining the Eq. (6), as follows:

$$f(n)=n+\log\ n^{*}cn \tag{6}$$

Therefore, the time complexity of Algorithm 1 satisfies the relationship $T(n)=O(f(n))=O(nlog(n))$; because calling for function needs to be carried out in the system stack, and maximum depth of the system stack is recursive call of the Cal-ET function. Two times later, the complexity space of the recursive algorithm is $O(nlog(n))$.

In Algorithm 3, running time is mainly spent on the "for" loop. It scans each coding symbol of the gene coding region from right to left. If the chromosome length is n and the eL is L, then the time complexity of algorithm is $O(L)$, and the maximum time complexity of algorithm is $O(n)$ while the gene coding region is the whole gene, where $L\leq n$. In Algorithm 3, the temporary space occupied while running depends largely on the size of operand stack. It is clear that its depth will not exceed the maximum number of terminals in the gene coding region (defined as i) of the chromosome. So it is space complexity is $O(i)$, and the number i of the terminals in the valid chromosome is always less or equal to the chromosome length n. Therefore, $O(i)<=O(L)<=O(n)$.

Evidently, compared with the Primitive GEP with tree structure resolution, the time complexity and space complexity of the S_GEP algorithm are decreased a lot in K-expression computing.

3.5. Method of construction ET

One of the biggest advantages of GEP inherited from GP is that individual can use the non-linear form (that is, expression trees) to denote complicated problems with stronger capability of resolving complex problems. In order to maintain problem-expressing ability of GEP, this study presents a new method of structure chromosome expression tree corresponding to the S_GEP algorithm. This method is known as the pre-order building tree. The tree built in this way is known as the S-ET. The basic idea of the pre-order building tree algorithm is as follows: to read the characters in the corresponding genes one by one from left to right, the first character regarded as the tree root, and the remaining follow-up nodes structuring the sub-tree, as a branch node added to the parents, recursively building the sub- tree according to priority on the depth from left to right until all the sub- trees becoming leaves. And the number of each sub-tree branches of every root is the number of arguments of the root's corresponding function with the terminals only as the leaves. For example, as shown in the Eq. (3) denoting ET of the traditional chromosome GEP and S-ET of the pre-order building tree shown in Fig. 3a and b.

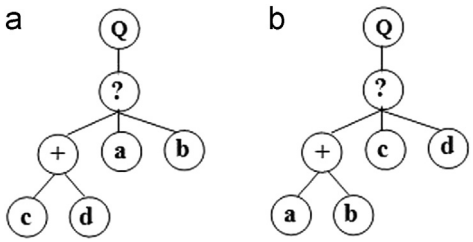

Fig. 3. (a) The ET of Eq.(3) built by traditional GEP. (b) The S-ET of Eq. (3) built by S_GEP.

To[0]	To[1]	To[2]	To[3]
a	b	c	d

Fs[0]	Fs [1]	Fs [2]
+	?	Q

Fig. 4. Process A, what described the process of decoding and evaluating chromosomes with S_GEP.

Theorem 2. *For any valid effective GEP genes, the pre-order traversing the gene S-ET, and the derived linear string in semantic meaning are grammar K-expressions.*

Proof. According to the basic idea of pre-order building tree algorithm, the linear string by pre-order traverse S-ET is a Polish notation. And as the construction process of S-ET denotes that in term of semantic any S-ET of effective GEP is on behalf of a mathematical or logical expression. Therefore, the linear string attain from pre-order traversing the S-ET of the gene must also be valid and grammar Polish notation in semantic. According to principles of GEP we know that this Polish notation is just the Open Reading Frame of the GEP gene; that is the K expression. □

Theorem 3. *For any effective GEP gene, the function expression attained from decoding and evaluating chromosomes with S_GEP and the function expression attained from the pre-order building tree method is equal.*

Proof. For any valid and effective GEP genes, if the process of decoding and evaluating chromosomes with S_GEP is recorded, preserving one by one in the array To[i] the data attained in popping stack sequence that the terminals corresponding numerical, and preserving the function with the array of Fs[h] scanned the gene by right-to-left, as Fig. 4 shows. According to the basic idea of S_GEP algorithm known that the process of the data value, which is corresponding to the terminal, popping stack is the process of the data value, which is corresponding to the terminal taking part in calculating, may wish to call this process A. Also, for any valid and effective GEP genes, if the process of scanning the expression tree built with pre-order building tree method, from left to right then bottom to up is recorded, preserving the value attained from the corresponding terminal one by one in the array Tt[i], preserving the function with the array of Ft[h], as Fig. 5 shows. Observed this process is that the gene S-ET take part in the corresponding node calculating in sequence, may wish to call this process B. Then, can be found To[i] and Tt[i], Fs[h] and Ft [h] are always the match. Obviously, the process A also match the process B, therefore, the expression formed by these two processes is also of the same, as Figs. 2 and 5 shows. □

So, according to Theorem 2 and 3 presented above, the ET construction method corresponding to the S_GEP is feasible.

4. Experiment and result analyses

In this section, several experiments using S_GEP for selected symbolic regression problems are designed to justify the advantage of S_GEP for symbolic regression problems. Two of these testing cases are real-world project problem, and other testing cases all have been studied by other researchers with regard to symbolic regression issue. Various experiments in this section are for two main purposes:(1) testing for improvements featured of S_GEP, and (2) testing S_GEP capability for both well-studied benchmark symbolic regression problem and real-world project problem. S_GEP and other GEP in all experiments reported in this paper are developed by C# 2.0, and run on a PC with Pentium4 2.4 GHz CPU, 1.0 GB memory and Windows XP professional sp2. In all experiments, 100 independent runs are performed to avoid stochastic deviation, and the stochastic tournament with scale 10% of population size is used for the selection method.

4.1. Test for improvements featured

These experiments aim to evaluate the improvements featured in S_GEP, as well as to compare its performance with the primitive GEP and GEP-RNC.

Tt [0]	Tt [1]	Tt [2]	Tt [3]
a	b	c	d

Ft[0]	Ft[1]	Ft[2]
+	?	Q

Fig. 5. Process B, what described the process of decoding and evaluating chromosomes with S_GEP.

Table 1
Planetary data to rediscover the Kepler's Third Law.

Planet	Distance	Orbital Period	Planet	Distance	Orbital Period
Venus	0.72	0.61	Jupiter	5.20	11.9
Earth	1.00	1.00	Saturn	9.53	29.4
Mars	1.52	1.84	Uranus	19.1	93.5

The first testing case is Kepler's Third Law, which is used in literature [20] as shown below Eq. (7), the testing data are shown in Table 1, to detail the efficiency improvement of individual evaluation method in S_GEP without any additional constant creating strategy (the spontaneous emergence of necessary constants is created through the evolutionary process of GEP), which is named as S_GEP_A hereafter. Five groups of contrasting experimental tests for this task and achieved good results as shown in Table 3.

$$P^2 = cD^3 \tag{7}$$

where P is the orbital period of a planet and D is the average distance from the Sun, c (set it as 1 in the experiments) is a constant.

The main parameters setting is shown in Table 2,

To simplify, we use abbreviation in Table 3. The MaxI means Max time for evaluating per individual, and the MinI means Min time for evaluating per individual, and the AverI means average time for evaluating per individual fitness.

In order to facilitate comparative analysis for the efficiency and capability of the proposed algorithms more intuitively, the more intensive statistical data counted from Table 3 are shown as Fig. 6 and Table 4.

The experimental results show that the S_GEP_A is superior to the primitive GEP: (1) Two algorithms are similarly successful as showed in Fig. 6 counted from Table 4. It proves the algorithm nature that the ability of expressing and solving problems of S_GEP_A is the same as that of the Primitive GEP algorithm; (2) Compared with the Primitive GEP, the S_GEP_A has obviously higher efficiency in the theory and experiments; (3) The experimental results above show that the more increase of population size, or the more increase of gene length, the more superiority of the S_GEP_A can be reflected. This is basically consistent with the preceding theory analysis.

The second and third testing case are designed to test the performance of our constant creating method and the efficiency of S_GEP, comparing with primitive GEP and GEP-RNC. Take the Sequence Induction (SI) problem and Quadratic Function Problem as cases, which are:

$$f(x) = 4x^4 + 3x^3 + 2x^2 + x \tag{8}$$

Table 2
Parameters experiments.

Parameters	Value	Parameters	Value	Parameters	Value
Function set	+−*/	Head length	12(17)	2-point cross rate	0.3
Terminal set	D	chomosome length	75(105)	1-point cross rate	0.4
Number of Gene	3	mutation rate	0.2	inversion rate	0.05
Link function	+	IS transposition rate	0.05		

Table 3
Results of compared experiment.

	Primitive GEP					S_GEP_A				
	Test1	Test2	Test3	Test4	Test5	Test1	Test2	Test3	Test4	Test5
Generation	100	100	100	100	1000	100	100	100	100	1000
Population size	100	100	200	300	200	100	100	200	300	200
Chromosome length	105	75	75	75	75	105	75	75	75	75
Success rate (%)	89	86	95	98	100	89	86	94	98	100
Runing time (s)	3.394	2.855	5.021	6.797	53.646	2.635	2.244	3.797	5.049	41.801
MaxI (s)	0.000414					0.000219				
MinI (s)	0.000081					0.000126				
AverI (s)	0.000252					0.000181				

$$f(x) = 2.718x^2 + 3.14163x \quad (9)$$

This time, the training dataset of Eq. (8) contains points computed for the first 10 positive integers. And the training dataset of Eq. (9) get from 10 random points, $x \in [-10,10]$.

The main parameters setting is the same as the previous experiment, and all methods evolve 5000 generations with 100 population size. Specially, this experiment set {*x*,1, 0.1} and 8 random float point as terminal set of S_GEP with constant terminal COR 0.2, {*x* } as terminal set of primitive GEP, {*x*,? } as terminal set of GEP-RNC with constant set length 10 and constant mutation rate 0.05. Because GEP-RNC uses the same gene head length 12 with S_GEP and primitive GEP, but it has a longer chromosome length 114 because of its Dc. Fitness function is *R*-square. We assume that when the fitness function achieves a value more than 0.9999, the optimal solution is successfully achieved. Experiment result is shown in Table 5. S_GEP has a strong capability of constant creation in two test cases with successful rates as 86% and 89%, and has bigger Average chromosome *R*-square in a run which means that it can significantly improve the fitness of average individuals in the population and find the optimal solution earlier. In addition, S_GEP operates faster than other GEP methods, although both of them evolve 5000 generations and with the same head length.

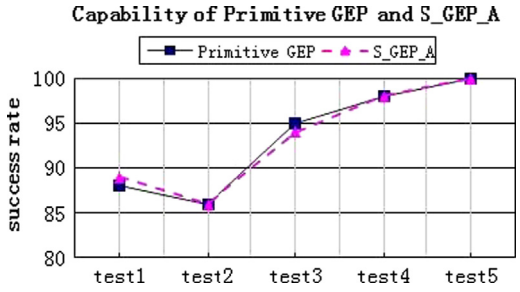

Fig.6. Ability comparing between the S_GEP_A and the Primitive GEP.

Table 4
Performance analysis for S_GEP_A.

Tested group	Number of generation/population size/chromosome length	Efficiency improving rate (%)
First	100/100/105	22.36
Second	100/100/75	21.40
Third	100/200/75	24.38
Fourth	100/300/75	25.72
Fifth	1000/200/75	22.08
Average fitness' time improving		28.17

As the experiment results shown in Table 5 above, regarding the success rate, S_GEP has a high success rate to solve this two well-studied benchmark problem, which is the same with the best of those of primitive GEP and GEP-RNC, but much better than that of GEP-RNC in Sequence Induction problem and that of primitive GEP in Quadratic Function Problem. Regarding the number of generations to find the optimal solutions, S_GEP consistently used less than that of primitive GEP in both two testing cases, and used less than that of GEP-RNC in Sequence Induction problem. Also, S_GEP needed much less computational effort than that of primitive GEP and GEP-RNC. Namely, S_GEP enhances the capability and raises the efficiency effectively.

4.2. Testing S_GEP capability for more symbolic regression problems

The following are four experiments aimed to verify the capability of S_GEP for other more symbolic regression problem compared with other methods.

4.2.1. Well-studied benchmark problem

Each GEP variant runs by using parameters simulating the testing for Sequence Induction problem and Quadratic Function Problem above. The performance of both approaches is compared in terms of average running time and average best-of-run fitness and average *R*-square. The results are shown in Table 6.

The first well-studied benchmark problem is a "V" shaped function was chosen:

$$f(a) = 4.251a^2 + \ln(a^2) + 7.243e^a \quad (10)$$

where *a* is the independent variable and *e* is the irrational number 2.71828183. Problems of this kind cannot be exactly solved by evolutionary algorithms [1].

The second well-studied benchmark problem is conducted predicting sunspots. In this case, 100 observations of the Wolfer sunspots series from 1700 to 1988, what were used for testing

Table 5
Performance of constant create and efficiency for S_GEP.

Index	Testing					
	Sequence Induction problem			Quadratic Function Problem		
	Primitive GEP	GEP-RNC	S_GEP	Primitive GEP	GEP-RNC	S_GEP
Success rate	90%	46%	89%	83%	88%	87%
Average generation of optimal solution	3687	4005	3479	3807	3356	3419
Average running time(s) each run	167	233	133	186	249	148
Average *R*-square of chomosome	0.8144	0.6152	0.8971	0.8096	0.7263	0.8842

Table 6
Performance of well-studied benchmark problem.

Index	Testing					
	"V" shaped function			Sunspots		
	Primitive GEP	GEP-RNC	S_GEP	Primitive GEP	GEP-RNC	S_GEP
Average time each run(s)	148	185	116	104	123	87
Average *R*-square	0.99643	0.95274	0.99912	0.81186	0.71337	0.88923
Average best-of-run fitness	1934.24	1913.82	1990.71	89033.29	86215.27	89736.01

several achine-learning systems research circles, are used with an embedding dimension of 10 and a delay time of one.

As the experimental results shown in Table 6 above, for two cases the quality of solutions found by S_GEP is better than those found by primitive GEP and GEP-RNC. Towards the same problems, the average computational effort required by S_GEP is much smaller than that of primitive GEP and GEP-RNC. It reveals the strong capability of S_GEP.

4.2.2. *Cunninghamia lanceolata plantation productivity problem*

The problem is a real-world problem of forestry using dataset from literature [22] to discover a function model (NPP) of cunninghamia lanceolata plantation annual average temperature productivity within China. This database has 50 samples with 5 variables, the same with the literature [22], we choose the most relevant annual average temperature variable for modeling. In order to compare with existing research conveniently, the quality of solutions is measured by a fitness function based on the Average *R*-square and the model precision (*P*) using in literature [23]:

$$P = \frac{1}{n}\sum_{i=1}^{n}\left(1-\left|\frac{NPP_i - P_i}{NPP_i}\right|\right) \tag{11}$$

And the compared result of the test is shown in Table 7. The curves of these algorithms result compared with original data are showed in Fig. 7, where, *t* is the annual average temperature variable.

In this case, as the results shown in the Table 7, the Average *R*-square and the model precision obtained by S_GEP is far beyond primitive GEP and method in literature [22]. Surely, the correlation model NPP curve obtained by S_GEP is more closer to the curve of original data than that obtained by primitive GEP and method in literatures [22,23], as shown in Fig. 7. It shows that S_GEP has a powerful potential to deal with real-world difficult nonlinear problems.

Table 7
Compared results of Npp testing.

Index	Method			
	LS[20]	Primitive GEP	RG-GEP[21]	S_GEP
***P*(%) average**	66.44	**81.34**	**82.74**	**88.07**
***R*-square**	0.7062	**0.8035**	**0.8862**	**0.9013**

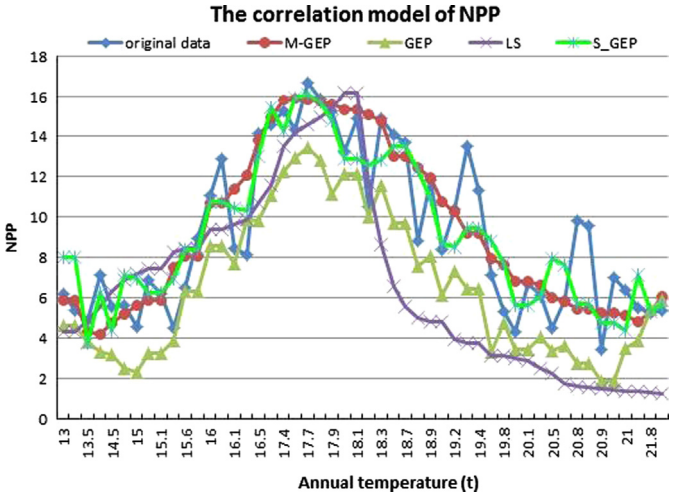

Fig.7. Results of cunninghamia lanceolata plantation productivity problem.

Table 8
Compared results of test for oak trees acorn size problem.

Index	Method		
	Primitive GEP	GEP-RNC	S_GEP
***P*(%)**	80.83	48.41	84.15
Average *R*-square	0.8435	0.5626	0.8831

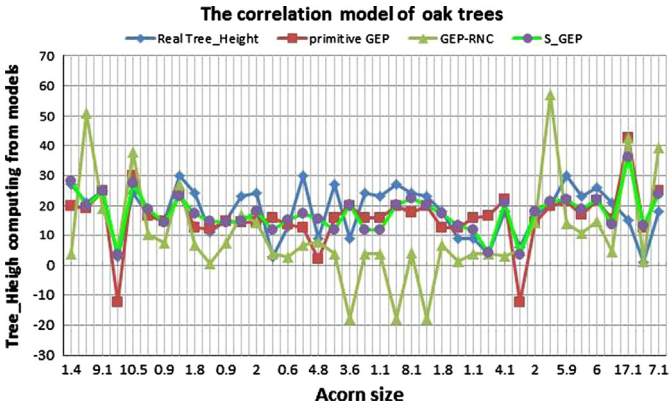

Fig.8. Symbolic regression results of oak trees acorn size problem.

4.2.3. *Oak trees acorn size problem*

This real-world problem is oak trees acorn size problem to find the relation function between oak trees acorn size and tree height within Atlantic and California, whose data are from http://lib.stat.cmu.edu, first reported in literature [24]. This database has 39 samples with 5 variable. In this study we choose only the acorn size and tree height for testing S_GEP. The parameters setting is the same with the testing for cunninghamia lanceolata plantation productivity problem previously. The results for this problem are shown in Table 8 and Fig. 8.

In this case, S_GEP and primitive GEP both have lower Average *R*-square and model precision rate than in cunninghamia lanceolata plantation productivity problem previously. Even so, overall, S_GEP reveals again a better performance than primitive GEP and GEP-RNC. The assumption that the quantity of data information has influence on the model quality is commonly made in nonlinear identification problems. Fewer samples in this case make the difficulty of the problem increased. Therefore, lower Average *R*-square and model precision rate are abtained in this case.

5. Conclusions

In this paper we present an improved gene expression programming (S_GEP) specially suitable for symbolic regression and we compare its performance against traditional GEP algorithms in several instances of symbolic regression problems. Besides, we experimentally clearly show that all improvements proposed in S_GEP over the primitive GEP and other related methods are advantageous. It not only can direct fast evaluate chromosomes based on the stack instead of building and then traversing the expression tree to save computational effort in the evolutionary process, but also can accelerate the convergence of the whole population to optimal. So, it can raise the capability and the efficiency for GEP.

Overall, the proposed S_GEP consistently better results than the system using traditional GEP. It shows a powerful potential of S_GEP to deal with complex nonlinear symbolic regression problems.

Surely, though our enhancements have been effective, there is still work that can be done to further improvement and promotion. Firstly, the impact of S_GEP operators setting on migration and the exchange of genetic material requires further study. In addition, the main contribution of this work consists in the improvements made in the basic gene expression programming algorithm first proposed in literature [1]. We understand that most of these improvements can be useful for other types of problems that can be solved in such an evolutionary computation technique. Hence, future work will focus on the adaptation and extension of S_GEP to other classes of problems.

Acknowledgments

We highly appreciate that this work is supported by the National Science Foundation of China Grant #60763012, and the National Science Foundation of Guangxi Grant #2012GXNSFBA053161, and the scientific research project of the Guangxi Education Department #201010LX293. Chang-an Yuan is the corresponding author.

Refrences

[1] C. Ferreira, Gene Expression Programming: a new adaptive algorithm for solving problems, Complex Syst. 13 (2) (2001) 87–129.
[2] C Zhou, W Xiao, P.C Nelson,and, T.M Tirpak, Evolving accurate and compact classification rules with Gene Expression Programming, IEEE Trans. Evolut. Comput. 7 (6) (2003) 519–531.
[3] V K Karakasis, A. Stafylopatis, Efficient evolution of accurate classification rules using a combination of gene expression programming and clonal selection, IEEE Trans. Evolut. Comput. 12 (6) (2008) 662–678.
[4] H.S. Lopes, W.R. Weinert, EGIPSYS: an enhanced Gene Expression Programming approach for symbolic regression problems, Int. J. Appl. Math. Comput. Sci. 14 (3) (2004) 375–384.
[5] C.A Yuan, C.J Tang, Y Wen, et al., Convergency of genetic regression in data mining based on Gene Expression Programming and optimized solution, Int. J. Comput. Appl. 28 (4) (2006) 359–366.
[6] A. Gandomi, S. Babanajad, A. Alavi, Y. Farnam, Novel approach to strength modeling of concrete under triaxial compression, J. Mater. Civ. Eng. 24 (9) (2012) 1132–1143.
[7] Zuo Jie, Tang Chang-jie, Li Chuan, et al., Time series prediction based on Gene Expression Programming[C], WAIM 3129 (2004) 56–64 (LNCS).
[8] YUAN Chang-an PENG Yu-zhong, et al., Multi-cellular Gene Expression Programming algorithm for function optimization, Control Theory Appl. 27 (11) (2010) 1585–1589.
[9] X. Li,C. Zhou,W. Xiao, P.C. Nelson, Prefix Gene Expression Programming[C], in: Genetic and Evolutionary Computation Conference (GECCO'05), Washington, DC., 2005.
[10] Lei Duan, Chang-jie Tang, et al., Design and implementation of ORF filter in Gene Expression Programming, J. Sichuan Univ. (Eng. Sci. Ed.) 39 (6) (2007) 102–106.
[11] B. Elena, B. Andrei, L. Henri, AdaGEP – an adaptive Gene Expression Programming algorithm[C], ninth international symposium on symbolic and numeric algorithms for scientific computing, IEEE Comput. Soc. (2008) 403–406.
[12] Noah Ryan, David Hibler, Robust Gene Expression Programming, Proced. Comput. Sci. 6 (2011) 165–170.
[13] Yu-zhong Peng, Chang-an Yuan,et al. SGDE-GEP: a novel algorithm of GEP, in: Proceedings of the 2008 International Conference on Computer Science and Software Engineering, 2008.
[14] ⟨http://www.symbolicregression.com/⟩.
[15] J.R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, Cambridge, MA, 1992.
[16] Wouter Minnebo, Sean Stijven, Empowering Knowledge Computing with Variable Selection, May 2011, ⟨http://styven.be/thesis/ThesisWouterSean_v2.pdf⟩.
[17] Candida Ferreira, Function finding and the creation of numerical constants in Gene Expression Programming. The Seventh Online World Conference on Soft Computing in Industrial Applications, 2002.
[18] Xin Li, Chi Zhou, Peter C. Nelson, Thomas M. Tirpak. Investigation of constant creation techniques in the context of Gene Expression Programming, Genetic and Evolutionary Computation Conference (GECCO-2004), June 26–30, 2004, Seattle, Washington, USA.
[19] A.K. Kordon Future trends in soft computing industrial applications, in: Proceedings of the 2006 IEEE Congress on Evolutionary Computation, 2006, pp. 7854–7861.
[20] D. Searson GPTIPS: Genetic Programming & Symbolic Regression for MATLAB, ⟨http://gptips.sourceforge.net⟩, 2009.
[21] Kaikuo Xu, Yintian Liu, Rong Tang, Jie Zuo, Jun Zhu, Changjie Tang. A novel method for real parameter optimization based on Gene Expression Programming, Appl. Soft Comput. J. (2008), http://dx.doi.org/10.1016/j.asoc.2008.09.007.
[22] Wen Yuanguang, Yuan Chang'an Liu Shirong, Distribution and simulation of actual productivity of *Cunninghamia lanceolata* plantation in China, Guangxi Sci. 95 (2) (1995) 56–62.
[23] Yao Yuan, Research and Application on Spatial Data Function Finding System Based on GEP[D] (Master thesis), Guangxi Teacher University, 2007.
[24] Meri O. Akwood, Enrique Jurado, Michelle Leishman, Mark Westoby, Geographic ranges of plant species in relation to dispersal morphology, growth form and diaspore weight, J. Biogeogr. 20 (5) (1993) 563–571.

YuZhong Peng received his master's degree in Computer Application Technology from Guangxi Teachers Education University, China, in 2009. He is an Associate Professor at Guangxi Teachers Education University. His research interests are in Evolutionary Computation and Data modeling.

Dr. ChangAn Yuan received the Ph.D. degree in Computer Application Technology from the Sichuan University, China, in 2006. He is a professor at Guangxi Teachers Education University. His research interests include Computational intelligence and Data mining.

Xiao Qin received her master's degree in Computer Application Technology from Guangxi Teachers Education University, China, in 2009. She is an Associate Professor at Guangxi Teachers Education University. Her research interests are in Evolutionary Computation and Data mining.

Dr. JiangTao Huang received his Ph.D. degree in Computer Application Technology from the Sichuan University, China, in 2011. He is an Associate Fellow at Guangxi Teachers Education University. His research interests are in Data mining and Information Fusion.

YaBing Shi received her master's degree in Computer Application Technology from Guangxi Teachers Education University, China, in 2009. She is a lecturer at Guangxi Teachers Education University. Her research interests are in Data mining.