Contents lists available at SciVerse ScienceDirect

Engineering Applications of Artificial Intelligence

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

Parse-matrix evolution for symbolic regression

Changtong Luo [a],*, Shao-Liang Zhang [b]

[a] Institute of Mechanics, Chinese Academy of Sciences, Beijing 100190, China
[b] Department of Computational Science and Engineering, Nagoya University, Nagoya 464-8603, Japan

ARTICLE INFO

Article history:
Received 24 February 2011
Received in revised form
31 March 2012
Accepted 16 May 2012
Available online 8 June 2012

Keywords:
Genetic programming
Data analysis
Symbolic regression
Grammatical evolution
Artificial intelligence
Evolutionary computation

ABSTRACT

Data-driven model is highly desirable for industrial data analysis in case the experimental model structure is unknown or wrong, or the concerned system has changed. Symbolic regression is a useful method to construct the data-driven model (regression equation). Existing algorithms for symbolic regression such as genetic programming and grammatical evolution are difficult to use due to their special target programming language (i.e., LISP) or additional function parsing process. In this paper, a new evolutionary algorithm, parse-matrix evolution (PME), for symbolic regression is proposed. A chromosome in PME is a parse-matrix with integer entries. The mapping process from the chromosome to the regression equation is based on a mapping table. PME can easily be implemented in any programming language and free to control. Furthermore, it does not need any additional function parsing process. Numerical results show that PME can solve the symbolic regression problems effectively.

1. Introduction

Symbolic regression aims at finding a symbolic, mathematical model that can describe and predict a given system based on observed input-response data. It plays an increasingly important role in many engineering applications including structural mechanics (Yang et al., 2005), signal processing (Yao and Lin, 2009), system identification (Patelli and Ferariu, 2010), industrial data analysis (Vladislavleva et al., 2010), etc.

As is well known, regression analysis (Chatterjee and Hadi, 2006) is a statistical technique for the modelling and analysis of numeric input-response data. Conventional linear/nonlinear regression is based on a certain model structure and optimizes the coefficients in the model, which can be described as follows:

$$\alpha^* = \arg\min_{\alpha \in \mathbb{R}^k} \sum_i \| f(\boldsymbol{x}^{(i)}, \alpha) - y_i \|,$$

where $\boldsymbol{x}^{(i)} \in \mathbb{R}^d, y_i \in \mathbb{R}$ are numeric input-response data, and the model function $f : \mathbb{R}^{d+k} \mapsto \mathbb{R}$.

However, if the model structure is unknown or wrong, or the concerned system has changed and the old model becomes untrustful, conventional regression does not work any more. In these cases, data-driven models are very helpful and desirable. Fortunately, symbolic regression can solve such problems (Vladislavleva et al., 2009). Symbolic regression can optimize the model structure and coefficients simultaneously. The objective of symbolic regression is to find an appropriate model from a space of all possible expressions $\mathcal{S}$ defined by a set of given operations (e.g., +, -, ∗, /, etc.) and functions (e.g., sin(), cos(),exp(),ln(), etc.), which can be described as follows:

$$f^* = \arg\min_{f \in \mathcal{S}} \sum_i \| f(\boldsymbol{x}^{(i)}) - y_i \|.$$

Genetic programming (GP) (Koza, 1992) is a classical method for symbolic regression. The chromosome of GP is the parse tree of an actual program. This tree-based representation makes GP difficult (although not impossible) to be implemented in general-purpose programming languages such as C/C++ and Fortran. As a result, GP is originally implemented in LISP (Goldberg, 1996), which is a functional programming language. Most of us are more familiar with and prefer to use the general-purpose programming languages. Consequently, GP's applications are limited.

To overcome this difficulty, O'Neill and Ryan (2001) proposed a grammatical evolution (GE). GE introduced a binary string for its chromosome presentation. This is the essential difference between GE and the classical GP. Due to the string-based representation, GE can be implemented in any programming language. However, GE is still not so easy to use. First, the implementation of GE is complicated because it needs an additional function parser for the encoding and decoding process. Next, the incomplete mapping and extra codons problems (O'Neill and Ryan, 2001; O'Neill and Brabazon, 2006) are common but difficult to handle.

* Corresponding author.
E-mail addresses: luo@imech.ac.cn (C. Luo), zhang@na.cse.nagoya-u.ac.jp (S.-L. Zhang).

0952-1976/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.engappai.2012.05.015

GE is not easy to programme without an external function parser and difficult to control without a careful treatment of incomplete mapping and extra codons. These difficulties are caused by its chromosome representation. In fact, GE uses a binary string to represent an individual. The chromosome is a one-dimensional string, which can be regarded as a highly compressed parse tree. In our view, GE has over compressed the parse tree, and useful information might be lost due to this compression. The decoding process of GE becomes not straightforward any more. Therefore, the difficulties (including incomplete mapping and extra codons) arise.

To keep more useful information, we propose a less compressed parse tree called parse-matrix in this paper. It uses a two-dimensional matrix with integer entries for its chromosome representation.

The two-dimensional matrix with integer entries can carry more information than the one-dimensional binary string (used in GE). Consequently, the decoding process (of mapping from the parse-matrix to its corresponding expression) are simplified and becomes straightforward. The evolutionary algorithm that uses the parse-matrix representation is referred to as parse-matrix evolution (PME). PME is very easy to use and free to control. It does not need any special target programming language or additional function parser. Numerical experiments show that PME can solve the symbolic regression problems effectively.

2. Chromosome of genetic programming

Chromosome presentation is the fundamental and most important element to be addressed in genetic programming. Besides the above mentioned Koza's tree-based and O'Neill's string-based presentations, linear-based and graph-based presentations are also widely used. Graph-based presentation is a natural extension to the tree-based one (Trees are special types of graphs). Several graph-based GPs including parallel distributed genetic programming (PDGP) (Poli, 1996) and Genetic Network Programming (GNP) (Mabu et al., 2007) have been proposed. They use directed edges to connect the nodes. This structure enables them to re-use the subtree evaluations for those are repeatedly emerged in an individual. In order to make a better use of computer architectures (computer programs are represented in a linear fashion) and avoid the use of computationally expensive interpreters or compilers, linear genetic programming (LGP) is proposed (Poli et al., 2008; Brameier and Banzhaf, 2007). In LGP, a chromosome is represented as a sequence of instructions.

Only tree-based and string-based representations are closely related to our new proposal and will be discussed in detail as follows.

2.1. Tree-based chromosome representation

In canonical genetic programming (GP), an individual is represented as a parse tree, in which every internal node has an operator/function and every leaf node has an operand. For example, the analytical expression $\sin(x_1^2-x_2)+\ln(x_1+x_2)$ can be represented as the parse tree shown in Fig. 1. The crossover between two individuals is applied by exchanging their subtrees. The mutation of an individual is applied by replacing a randomly chosen subtree by another randomly generated subtree. Because of its tree-based representation, GP is easy to be implemented in functional programming languages (Goldberg, 1996) (e.g., LISP), but difficult to be coded in general-purpose programming languages such as C/C++ and Fortran. Obviously, most people know about general-purpose programming languages better and prefer using it if possible. There are a number of C/C++/JAVA

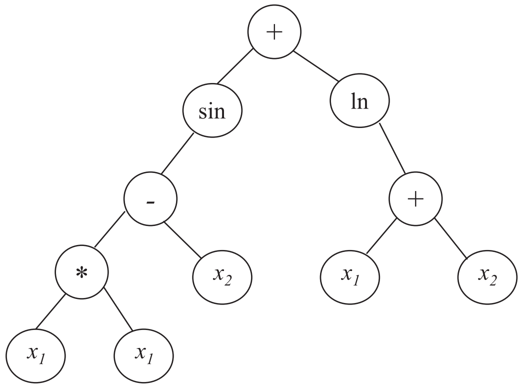

Fig. 1. An example parse tree: $\sin(x_1^2-x_2)+\ln(x_1+x_2)$.

implementations of GP like Lil-GP, GPC++, and DGPF (See Wikipedia item *Genetic programming* at http://en.wikipedia.org/wiki/Genetic_programming). Although it may be not so difficult to use any one of them as a standalone program, none of them is easy to be adapted for your own needs because of its complexity.

2.2. String-based chromosome representation

Grammatical evolution (GE) (O'Neill and Ryan, 2001) is a new variant of genetic programming, it uses a variable-length binary string to describe an individual. The binary string (genotype) has the following form:

$I=b_1b_2b_3\cdots b_M$, where $b_k\in\{0,1\}$.

GE decodes its chromosome as follows. First, the binary string is converted to an integer string of the form:

$I=[i_1,i_2,\ldots,i_{M/8}]$,

where each integer entry i_k maps to an 8-bit segment in the binary string. Next, the integer string is decoded component-wise to a derivation tree using a Backus-Naur form (BNF) grammar. Then, the derivation tree is decoded to a parse tree. Finally, the parse tree will be parsed to an analytical expression (phenotype) by a function parser (see O'Neill and Brabazon, 2006 for more details).

The BNF grammar used in GE can be represented by the tuple $\{N, T, P, S\}$, where N is the set of nonterminals, T the set of terminals, P a set of production rules that maps the elements of N to T, and S is a start symbol that is a member of N. Now we give a simple example BNF grammar as follows:

(a) $N=\{\text{expr},\text{op}\}$;
(b) $T=\{+,-,*,x,y\}$;
(c) $S=\langle\text{expr}\rangle$;
(d) the production rules P satisfies:

$$\begin{array}{lll}
\langle\text{expr}\rangle ::= & \langle\text{expr}\rangle\langle\text{op}\rangle\langle\text{expr}\rangle & (0)\\
 & |\langle\text{var}\rangle & (1)\\
\langle\text{op}\rangle ::= & + & (0)\\
 & |- & (1)\\
 & |* & (2)\\
\langle\text{var}\rangle ::= & x & (0)\\
 & |y & (1)
\end{array}$$

Suppose we have a chromosome $I=[88, 211, 8, 35\,133, 81, 68]$, and we want to decode it based on the above BNF grammar, then the decoding process can be listed in Table 1. We can see that the phenotype of I is $x*y$. Table 1 shows that the decoding process

Table 1
Decoding process of the chromosome $I=[88, 211, 8, 35, 133, 81, 68]$ using the example BNF grammar.

Step	Codon	Decoding	No. of choice	Operation	Choice	Resulting string
0						$\langle$expr$\rangle$
1	88	$\langle$expr$\rangle$	2	88 mod 2=0	$\langle$expr$\rangle\langle$op$\rangle\langle$expr$\rangle$	$\langle$expr$\rangle\langle$op$\rangle\langle$expr$\rangle$
2	211	$\langle$expr$\rangle$	2	211 mod 2=1	$\langle$var$\rangle$	$\langle$var$\rangle\langle$op$\rangle\langle$expr$\rangle$
3	8	$\langle$var$\rangle$	2	8 mod 2=0	x	$x\langle$op$\rangle\langle$expr$\rangle$
4	35	$\langle$op$\rangle$	3	35 mod 3=2	$*$	$x*\langle$expr$\rangle$
5	133	$\langle$expr$\rangle$	2	133 mod 2=1	$\langle$var$\rangle$	$x*\langle$var$\rangle$
6	81	$\langle$var$\rangle$	2	81 mod 2=1	y	$x*y$
	68					

terminates until the codon '81'. As a result, the last codon '68' is an extra codon. Obviously, if the chromosome I has more codons besides the listed codons, e.g., $I=[88, 211, 8, 35, 133, 81, 68, 54,108, 36, \ldots]$, all codons after the codon '81' will be regarded as extra codons and ignored by the function parser. On the contrary, if the chromosome I has less codons, e.g., $I=[88, 211, 8, 35, 133]$, we will get $x*\langle$var$\rangle$, which is an illegal analytical expression. In this case, the decoding process is incomplete.

GE uses a string-based chromosome. The string is one-dimensional, and it can be regarded as a highly compressed parse tree. This linearized non-tree representation makes GE easier to implement in general-purpose programming languages than classical GP.

However, GE is still not so easy to use. As can be seen from the above example, GE's mapping process from the genotype (a binary string) to the phenotype (an expression) is rather complicated. Consequently, GE needs an additional function parser for the decoding process. We can also see that the incomplete mapping and extra codons are common problems during the decoding process, which make GE harder to control.

In our view, GE has encountered these difficulties because it has over compressed the parse tree, and some useful information is lost due to the over-compression. This motivates us to design a less compressed, two-dimensional, non-tree chromosome representation, which has a simpler decoding process. The proposed chromosome representation called parse-matrix. It represents an individual as a two-dimensional matrix with integer entries. Obviously, it contains more information than one-dimensional binary string. Detailed description of parse-matrix will be presented in the next subsection.

Recently, McKay et al. (2010) present a survey of grammar-based genetic programming. They summarized different ways of chromosome representation, and classified them into several groups: tree-based grammar, linearized grammar, semantic grammars, logic grammars, tree adjoining grammar, etc. We confine our discussion to the tree-based and linearized grammar in this paper because these two chromosome representations are closely related to our proposal.

2.3. Parse-matrix chromosome representation

The new proposed chromosome representation, parse-matrix, is also derived from the parse-tree of genetic programming. First, let us take a deep look at the parse tree of GP and its evolutionary process. Usually, a parse tree consists of many subtrees. These subtrees can be classified into different levels. For example, the parse tree in Fig. 1 has four levels of subtrees, which is denoted by $S^{(1)},S^{(2)},S^{(3)},S^{(4)}$ (see Fig. 2). Higher level subtrees are produced from lower level subtrees.

For general symbolic regression problems, there are two kinds of operators – unary operators $\mathcal{U}$ (such as sin(), cos(), exp(), ln(), etc.), and binary operators $\mathcal{B}$ (such as +, -, $*$, /, etc.). Let the set of operators at internal nodes be $\mathcal{T}=\{+,-,*,/,\sin(),\cos(),\exp(), \ln(),\text{etc.}\}$. For the sake of simplicity, we define $T(s_1,s_2)$ as follows to unify the two kinds of operators:

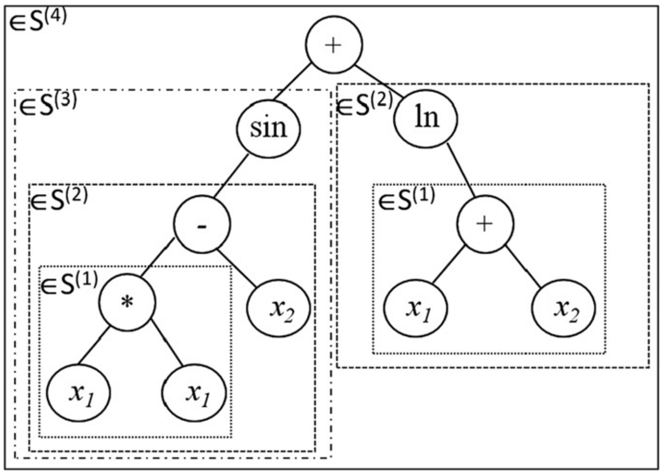

Fig. 2. Different level subtrees of the parse tree shown in Fig. 1.

$$T(s_1,s_2)=\begin{cases} T(s_1,s_2) & \text{if } T\in\mathcal{B};\\ T(s_1) & \text{if } T\in\mathcal{U}.\end{cases}$$

Let $S^{(0)}$ be the set of terminal expressions (level zero subtrees), e.g., $S^{(0)}=\{1.0,x_1,x_2,\ldots,x_d\}$, and $S^{(1)}$ be the set of all possible subtrees produced by the operators in $\mathcal{T}$ with terminals from $S^{(0)}$, then $S^{(2)}$ is the set of all possible subtrees produced by the operators in $\mathcal{T}$ with subtrees from $S^{(0)}\cup S^{(1)}$. Similarly, $S^{(i+1)}$ is the set of all possible subtrees produced by the operators in $\mathcal{T}$ with subtrees from $S^{(0)}\cup S^{(1)}\cup\cdots\cup S^{(i)}$, and so on and so forth. Suppose the highest-level subtrees be $S^{(K)}$, the objective of symbolic regression is to find an optimal tree (represents an optimal expression) within all level sets of subtrees $S^{(0)}\cup S^{(1)}\cup\cdots\cup S^{(K)}$. In other words, symbolic regression needs to find an optimal function f^*, such that

$$f^*=\arg\min_{f\in S^{(0)}\cup S^{(1)}\cup\cdots\cup S^{(K)}}\sum_i \|f(\boldsymbol{x}^{(i)})-y_i\|,$$

where $\boldsymbol{x}^{(i)}\in\mathbb{R}^n, y_i\in\mathbb{R}$ are numeric input-response data.

We can see that there are inclusion relationships between the sets of different level subtrees. First, we have $S^{(0)}\subset S^{(1)}$. In fact, $\forall s\in S^{(0)}$, $\exists s\in S^{(1)}$, s.t. $s=1.0*s$. Since $S^{(2)}$ is generated from $S^{(0)}\cup S^{(1)}$, and $S^{(0)}\subset S^{(1)}$, we can also say that $S^{(2)}$ is generated from $S^{(1)}$. Similarly, we have $S^{(i-1)}\subset S^{(i)}$, and $S^{(0)}\cup S^{(1)}\cup\cdots\cup S^{(i)}=S^{(i)}$. So the optimal tree can be found within the set of top level subtrees $S^{(K)}$. Based on this fact, we can define a parse-matrix to represent the set of top level subtrees. Thus, the objective of symbolic regression becomes to find an optimal parse matrix which maps to the optimal function f^*. To store the lower level subtrees for reuse, intermediate expressions such as f_1, f_2 are also introduced.

Based on the above discussion, we introduce a matrix with four columns of the form

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m3} & a_{m3} & a_{m4} \end{pmatrix}$$

to represent an individual (or 'chromosome') , where the entries a_{ij} are bounded integers near zero. The first column $a_{\cdot 1}$ indicates the active operator from set $\mathcal{T}$. The 2nd and 3rd columns $a_{\cdot 2}$,$a_{\cdot 3}$ indicate the active operands. The 4th column $a_{\cdot 4}$ decides which intermediate expression (f_1 or f_2) will be updated.

3. Parse-matrix evolution

The evolutionary algorithm that uses parse-matrix (see Section 2.3) as its chromosome presentation is referred to as parse-matrix evolution (PME) in this paper.

PME does not use the BNF grammar for its encoding or decoding. Alternatively, it uses a table of mapping rules which is much easier to understand and use. Table 2 gives an example mapping table, where d is the dimension of the target model. We will use this table to do our numerical experiments in the following section (see Section 6). Of course you can define your own table of mapping rules according to different problems.

3.1. Encoding of PME

The phenotype of PME (i.e., the target function of symbolic regression) is an analytical function/expression. The encoding from the phenotype to the genotype (a parse-matrix) is straightforward. For example, the expression $\sin(x_1^2 - x_2) + \ln(x_1 + x_2)$ can be produced in the steps listed in Table 3.

According to the mapping rules in Table 2 and the encoding steps in Table 3, the expression $\sin(x_1^2 - x_2) + \ln(x_1 + x_2)$ can be described by the parse-matrix as follows:

$$A = \begin{pmatrix} 2 & 1 & 1 & 0 \\ 1 & 1 & 2 & 1 \\ -1 & -1 & 2 & 0 \\ -4 & -2 & 1 & 1 \\ 3 & -1 & 2 & -1 \\ 1 & -3 & -2 & 0 \end{pmatrix}.$$

Table 2
An example mapping table for parse-matrix evolution.

$a_{\cdot 1}$	-4	-3	-2	-1	0	1	2	3	4
T	ln	cos	/	–	skip	+	*	sin	exp
$a_{\cdot 2}$,$a_{\cdot 3}$	-3	-2	-1	0	1	2	$\cdots$	d	
expr	f	f_2	f_1	1.0	x_1	x_2	$\cdots$	x_d	
$a_{\cdot 4}$	-1	0	1						
$f \to f_k$	Skip	f_1	f_2						

Table 3
Encoding steps of the expression $\sin(x_1^2 - x_2) + \ln(x_1 + x_2)$.

Step	T	s_1	s_2	Which one is to be updated: f_1 or f_2?
1	*	x_1	x_1	Update f_1
2	+	x_1	x_2	Update f_2
3	–	f_1	x_2	Update f_1
4	ln	f_2	x_1	Update f_2
5	sin	f_1	x_2	None
6	+	f	f_2	Update f_1

We can see that the encoding of PME is a natural and easy process. Note that it is hard to imagine how to encode an expression to a binary string in GE.

3.2. Decoding of PME

The genotype of an individual (a chromosome) in PME is a parse-matrix. Its corresponding phenotype is an expression (a function). A parse-matrix will be decoded according to the mapping rules listed in Table 2 row by row as follows.

Procedure of decoding:

```
for ($i=0$; $i<m$; $i$++){
  determine operands $s_1$ and $s_2$ according to $a_{\cdot 2}$ and $a_{\cdot 3}$;
  switch ($T$){
    case $a_{\cdot 1}$: $f=T(s_1, s_2)$;
    update $f_1$ or $f_2$ according to $a_{\cdot 4}$;
  }
}
```

Suppose we have an individual (a parse-matrix, the genotype)

$$A = \begin{pmatrix} 2 & 1 & 2 & 0 \\ 3 & -1 & 1 & 1 \\ 1 & 1 & -2 & -1 \\ 0 & -1 & -2 & 0 \\ -4 & -3 & 2 & 0 \\ 0 & 2 & -1 & 1 \end{pmatrix},$$

and we want to get its phenotype. The decoding process can be listed in Table 4.

From Table 4, we can see that the decoding process is also natural and easy.

Although the mapping between the set of parse matrices and the set of symbolic expressions is not one-to-one, the interpretation between them is very straightforward and easy. We can see from the above examples that the coding plan (encoding and decoding) of PME is almost invertible. This is a favourable feature. As we know that it is almost impossible to reduce a symbolic expression to its original binary string in GE. Furthermore, GE's decoding from a binary string to its phenotype (a symbolic expression) is also not so easy. For example, the incomplete mapping might result in an illegal analytical expression; the extra codons often confuse users, and you could not predict where the first extra codon occurs unless you decode it bit by bit and find that the decoding process comes to an end. This makes GE hard to control.

Remark A. The height of the parse-matrix m reflects the level of top subtree K. It is easy to see that $K \le m$. As a result, m can be used to control the depth of searching process.

Remark B. Provided f_1,f_2,f are properly initialized (e.g., set $f_1 = f_2 = f = 1$), every row of the parse-matrix maps to a complete operation and updating process. Thus, there is no incomplete mapping problem in PME.

Table 4
Decoding steps of the individual A.

Step	f_1	f_2	f
1	$x_1 * x_2$	x_1	$x_1 * x_2$
2	$x_1 * x_2$	$\sin(x_1 * x_2)$	$\sin(x_1 * x_2)$
3	$x_1 * x_2$	$\sin(x_1 * x_2)$	$x_1 + \sin(x_1 * x_2)$
4	Nothing changed		
5	$x_1 + \sin(x_1 * x_2)$	$\sin(x_1 * x_2)$	$\ln(x_1 + \sin(x_1 * x_2))$
6	Nothing changed		

Remark C. Only unary and binary operators are considered here. As a result, the column number of the parse-matrix equals to 4. However, parse-matrix could easily be extended so that it can cope with multiple operators (e.g., the scalar triple product $\boldsymbol{a}\cdot(\boldsymbol{b}\times\boldsymbol{c})$). In general, suppose the operator T has p operands, we can just increase the column number to $p+2$ and define

$$T(s_1,\ldots,s_p)=\begin{cases} T(s_1) & \text{if } T\in\mathcal{U};\\ \mathrm{T}(s_1,s_2) & \text{if } T\in\mathcal{B};\\ \cdots, & \cdots;\\ T(s_1,\ldots,s_p) & \text{if } T \text{ has } p \text{ operands}.\end{cases}$$

then the operators with different number of operands can be unified.

3.3. Biological background

The representation of PME is inspired from the structure of genetic system (Hartl and Jones, 2008). In fact, the meaning of a parse-matrix in PME may be compared to a chromosome in biology (see Fig. 3):

1. An element of the parse-matrix itself has no explicit meaning. It functions like a base in a codon.
2. Every row of the parse-matrix maps to a complete operation and updating process. It functions like a four-base codon.
3. The whole parse-matrix carries all information of an expression. It plays the role of a chromosome.

3.4. Evolutionary operators

PME uses a parse-matrix for its chromosome representation, and a table of mapping rules for its encoding and decoding. Note that each row of the matrix represents a complete operation and updating process. Therefore, it is very easy to design evolutionary operators for PME.

- *Crossover*: Traditional crossover is easy to be adapted for PME. For example, we can define a one-point crossover as follows (see Fig. 4(a)): Select a cross-point r_c from $\{1,2,\ldots,m-1\}$ at random, and exchange the last $m-r_c$ rows of two parents of parent matrices. We can also define a two-point crossover (see Fig. 4(b)) that the two parents exchange their middle rows between r_1 and r_2, where $r_1,r_2\in 1,2,\ldots,m-1$ are two cross-over points such that $r_1<r_2$. A cut-and-splice crossover (see Fig. 4(c)) might also be used, where the first r_1 rows of parent A and the first r_2 rows of parent B are exchanged. We can see that the offsprings generated by any of the three kind of crossovers are valid if and only if their parents are valid (because each row of the parse-matrix represents a complete operation and updating process). That is, there is no incomplete mapping issue.
- *Mutation*: The mutation operator is also easy to design. For example, a uniform mutation (see Fig. 4(d)) can be defined as follows. Generate a random number $r_{ij}\in[0,1]$ for each entry of the parse-matrix to decide whether to mutate a_{ij} or not (according to the preset mutation probability P_m). The mutated entry a'_{ij} will be randomly chosen from the set $a_{\cdot j}\backslash\{a_{ij}\}$, where $a_{\cdot j}$ denotes the set of all possible values of the j-th column as listed in Table 2.

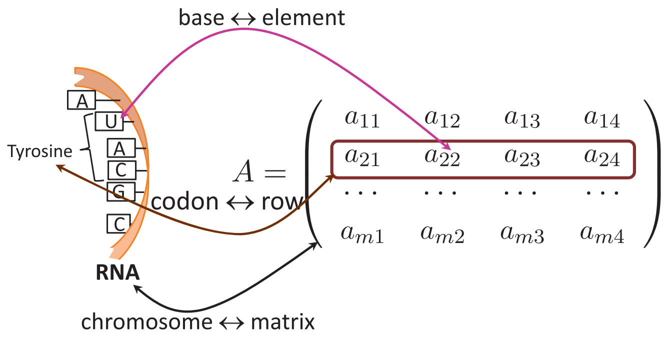
base ↔ element
A
U
A
C
G
C
Tyrosine
codon ↔ row
RNA
chromosome ↔ matrix
$A=\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14}\\ a_{21} & a_{22} & a_{23} & a_{24}\\ \cdots & \cdots & \cdots & \cdots\\ a_{m1} & a_{m2} & a_{m3} & a_{m4}\end{pmatrix}$

Fig. 3. Biological interpretation of parse-matrix.

3.5. Search engine

Suppose that the dimension of the target system is d, the height of the parse-matrix m, and we use the mapping Table 2, then the parse-matrix entries $a_{\cdot 1}\in\{-4,-3,\ldots,3,4\}$, $a_{\cdot j}\in\{-3,-2,\ldots,d\}(j=2,3)$, and $a_{\cdot 4}\in\{-1,0,1\}$. Thus there are $(9\cdot(4+d)\cdot(4+d)\cdot 3)^m$ combinations of the parse-matrix $(a_{i,j})_{m\times 4}$. If $d=2$, $m=4$, we have

$$(9\cdot(4+d)\cdot(4+d)\cdot 3)^m=2.8\times 10^{12}.$$

We can see that the search space of PME is very large in general. Therefore, enumeration method is impractical and effective search engines are necessary.

Fortunately, many state-of-the-art search engines are still applicable to PME. In fact, PME uses a parse-matrix as its chromosome presentation, and the evolutionary operators (crossover and mutation) are slightly modified from traditional ones. Therefore, PME does not require any special search engine. For example, the classic genetic algorithm (GA) with roulette-wheel selection or the steady-state GA (Blickle and Thiele, 1996) with tournament selection could easily be adapted to PME. In this paper, we choose the steady-state GA as PME's search engine to do our numerical experiments because it is very easy to implement and it has less stochastic noise.

3.6. Procedure

Despite different search engines might be used, the general procedure of PME could be described as follows.

Procedure of PME:

Step 1 : Input evolutionary parameters: population size N, initial bounds of entries for each columns $l,u\in\mathbb{Z}^4$, and set $t:=0$; Initialize the current and intermediate expressions f,f_1 and f_2.

Step 2 : Initialize population $\overrightarrow{A}(t)=\{A_1(t),A_2(t),\ldots,A_N(t)\}$ and evaluate each individual of the current population $\overrightarrow{A}(t)$, where the individual $A_i(t)$ is a matrix in $\mathbb{Z}^{m\times 4}$;

Step 3 : Repeat the following steps until some stopping criterion is satisfied.

(3.1) Reproduction: Apply the crossover and mutation operators to generate a trial population $\overrightarrow{A}'(t)$ based on the current population $\overrightarrow{A}(t)$ (see Section 3.4).

(3.2) Decoding: Decode each individual of the trial population (see Section 3.2);

(3.3) Evaluation: Evaluate each individual $A'_i(t)$ of the new population $\overrightarrow{A}'(t)$ according to the residual between its represented model and the exact model:

$$r=\sum_i\|f_{A'_i(t)}(\boldsymbol{x}^{(i)})-y_i\|;$$

(3.4) Selection: Select offspring individuals for next generation $\overrightarrow{A}(t+1)$, i.e., select N individuals among the current population $\overrightarrow{A}(t)$ and the trial population $\overrightarrow{A}'(t)$.

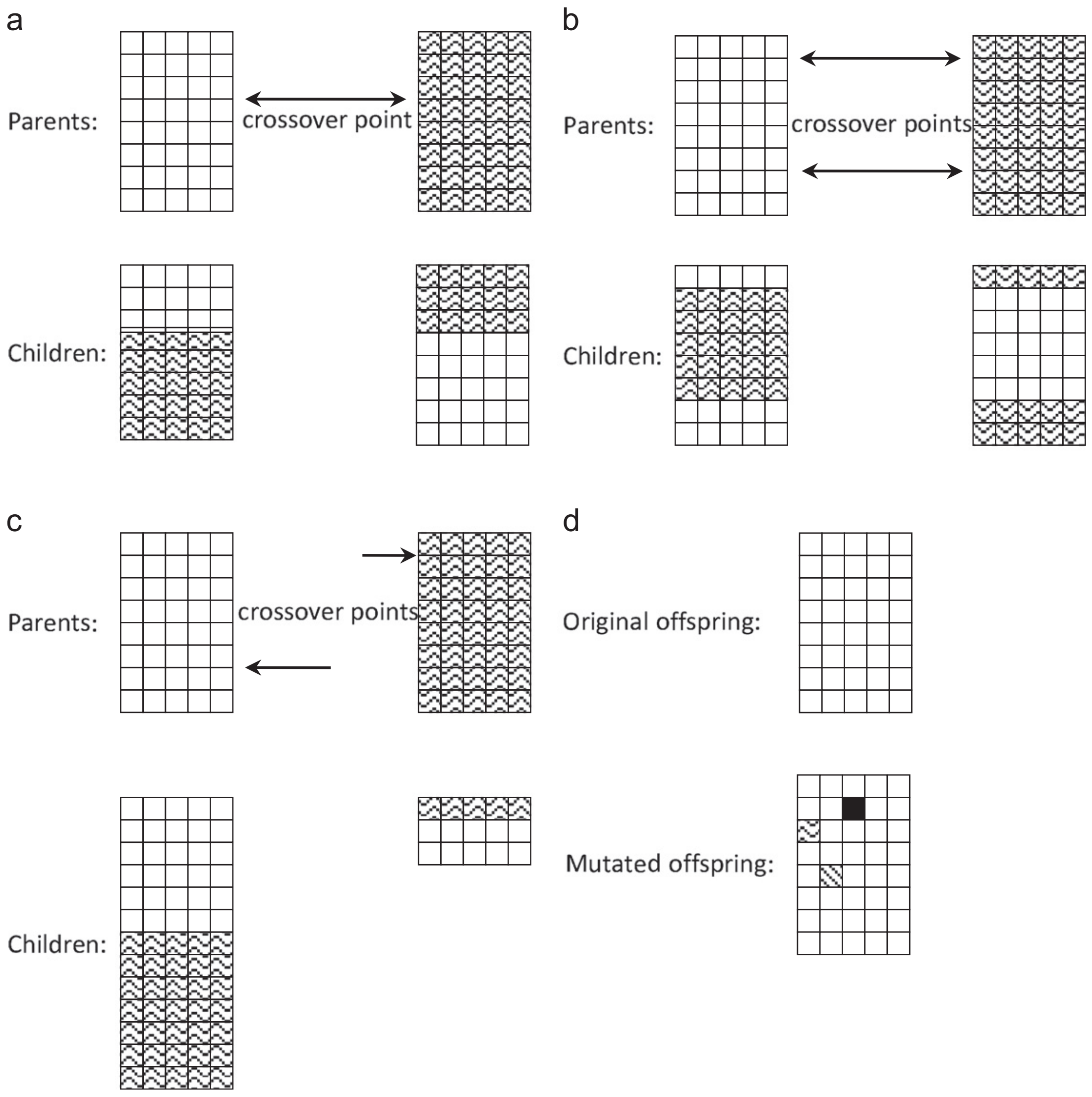

Fig. 4. Possible reproduction operators. (a) One-point Crossover, (b) Two-point Crossover, (c) Cut-and-splice Crossover and (d) Uniform Mutation.

(3.5) Check point: If some stopping criterion is satisfied, output the best-so-far individual A^* and its function value $f(X^*)$; Otherwise, set the current generation $t := t+1$, set the current position $i := 1$ and then return to step 3.

3.7. Exceptions

Note that infeasible individuals might be generated due to the existence of logarithmic (log()) and divisional (/) operators. An individual is said to be infeasible if its expression contains a logarithmic operator and the argument is negative or zero at some trial points, or its expression contains a divisional operator and the denominator happens to be zero at some trial points.

In our implement of PME, infeasible individuals are differently treated at different evolutionary stages. At the initialization stage, an infeasible individuals will be replaced by a random-generated feasible individual. As a result, all individuals in the initial population will be feasible. During the reproduction stage, once an infeasible individual is detected, it will be labelled as a looser and its fitness will be defined as zero. Consequently it cannot survive to the next generation.

4. Remarks for engineering applications

4.1. Practical difficulties

Practical engineering problems are hard. Calculation of the response can be computationally/experimentally expensive. The design variables are not necessarily independent. The input-response data might be high dimensional, not designed, noisy, redundant, sparse, and/or with multiple time scales. They may cause many difficulties to genetic programming (including PME) for extracting valuable information from the data. Over-fitting, bloat, premature convergence are 'classical' difficulties in the GP field. Many problems are beyond the scope of this paper, and will be addressed in future studies.

4.2. Complexity control

Let us compare the target functional spaces of conventional linear/nonlinear regression:

$$\alpha^* = \arg\min_{\alpha \in \mathbb{R}^k} \sum_i \| f(\boldsymbol{x}^{(i)}, \alpha) - y_i \|$$

and symbolic regression

$$f^* = \arg\min_{f \in \mathcal{S}} \sum_i \| f(\boldsymbol{x}^{(i)}) - y_i \|.$$

For the conventional linear/nonlinear regression, the target function space $\mathcal{F} = \{ f(\boldsymbol{x}, \alpha) | \alpha \in \mathbb{R}^k \}$. It is obviously that $\mathcal{F} \subset \mathcal{S}$. That is, symbolic regression has a larger target function space, and linear/nonlinear regression might be considered as a special case of symbolic regression. This is the reason why a symbolic regression algorithm is expected to get a better model than a linear/nonlinear regression method.

Beware that too large searching space might result in unreal models (over-fitting). This phenomenon has been detected in numerical analysis. A case in point is Runge's phenomenon, in which a problem of oscillation that occurs when interpolating with high degree polynomials. Such phenomenon might also appear if the symbolic regression uses high level parse trees. Therefore, it is very important to properly confine the target function to a subset of function space. If we can find the minimum space where the minimum complexity model lies, the symbolic regression becomes easier.

PME is designed easily to control the complexity of the evolved expression. The height of the parse matrix (m) in PME is an explicit control parameter that can upper-bound the subtree-level of evolved expression. So it can be used to confine PME's target function space. The parse tree will not bloat if we choose a fixed m (as we did in this work) or a controlled m, and the over-fitting difficulty is also avoided. It is easy for PME to balance the trade-off between the accuracy (model error) and complexity. In this sense, PME is free to control.

4.3. *Premature avoidance*

To avoid premature convergence, a multi-start technique (Martí, 2003) is used in PME. Every T generations the population is re-initialized. The archived best individuals remain survived separately from the population.

4.4. *Cost of symbolic regression is negligible*

We have to point out that symbolic regression is not the most expensive part to construct a data-driven model in practical applications. In fact, symbolic regression is just an analytical process. Usually, it only consumes CPU time from minutes to hours. But the training points are obtained by costly physical experiments or computational intensive simulation process. So it is much more expensive to get the set of training points (the input-response data). Therefore, it is very important for a symbolic regression algorithm to rebuild the best structure of the data-driven model at 'any' cost.

5. Coefficient optimization

The mission of symbolic regression is to optimize both the structure and coefficients of a target function that describes an input-response system. PME, as well as other symbolic regression algorithms, distinguishes from linear/nonlinear regression in that its capability of 'structure optimization', which is its true value.

We do have enabled PME capable of optimizing the structure and coefficients simultaneously, and it works well (see Section 6.2). But in our real-world application of ground-to-flight data correlation, we find that the revolved results are difficult to physically interpret and the physicist (decision maker) do not accept them. Therefore, we introduced some techniques to avoid the 'coefficient optimization', and only use PME's 'structure optimization' capability to carry out adaptive space transforms. The results become interpretable and acceptable. We conclude that PME is NOT just an alternative to symbolic modeling. It can be used to detect the underlying physical relationships between different parameters.

To enable PME's capability of coefficient optimization, we have re-designed the decoding and individual evaluation process. In the modified decoding process, coefficients are introduced, and during the individual evaluation process, a global optimization algorithm, low dimensional simplex evolution (LDSE) (Luo and Yu, 2012), is embedded to complete the online optimization of the introduced coefficients.

There are several possible ways to introduce coefficients into the evolving function.

(a) Using multi-gene techniques as in the GP software GPTIPS (Searson et al., 2010), in which the evolved functions are linear combinations of low order non-linear transformations of the input variables.
(b) Using the random numeric terminal as the reference (Streeter and Becker, 2003).
(c) Adding coefficients during the decoding process and optimizing them during the individual evaluation (our proposal).

We choose plan (c) in our work. Here is the reason: Plan (a) combines linear and non-linear optimization process, so the revolved model accuracy is likely high. But the result is hard to interpret. Plan (b) is very ease to implement and will not increase much computation cost, but the inherent randomness is hard to be accepted by the decision maker.

Symbolic regression is superior to linear/non-linear regression in that it can detect the desired function structure (usually non-linear) without making any priory assumptions on the target function. Plan (c) can hold this advantage.

In this work, coefficients are added in the following way: whenever the intermediate expression f_2 is updated (in case $a_{\cdot 4} = 1$ in Table 2), a new coefficient will be introduced and the intermediate expression will be further updated to $f_2 = \lambda_k f_2$, where k starts from 1 and $k \le m$.

For example, suppose we have two individuals from Sections 3.1 and 3.2 respectively, then the new decoding process will get $f = \sin(x_1 * (x_1) - x_2) + \lambda_2 * \ln(\lambda_1 * (x_1 + x_2))$ and $f = \ln(x_1 + \lambda_1 * \sin(x_1 * x_2))$ respectively.

Of course, one can introduce more intermediate expressions (e.g., $f_3, f_4, \cdots$) and attach coefficients whenever they are updated.

The modified PME process can be described as a bi-level optimization process of the following form.

$$\min_{A \in \mathbb{Z}^{m \times 4}} \min_{\lambda \in D \subset \mathbb{R}^m} \mathrm{MSE}(A, \lambda, x) = \frac{1}{K} \sum_{i=1}^{K} \| f_A(\boldsymbol{x}^{(i)}) - y_i \|_2$$

where MSE denotes the mean square error, A is the individual (parse matrix) of PME, f_A is the phenotype of A (an analytical function), λ is the coefficient vector, $\| \cdot \|_2$ denotes 2-norm.

The inner optimization (LDSE) will be invoked to optimize the coefficients in the model whenever an individual A is evaluated. The outer optimization (basic PME described in the previous sections) aims at optimizing the model structure. Their relations is shown in Fig. 5.

6. Numerical results

PME has been implemented in C programming language, using steady-stage GA with multi-start technique (Martí, 2003) as its search engine. For the sake of easy use, we use a Boolean variable

Outer optimization

Inner optimization

$$\boxed{\boxed{\mathrm{MSE}(A,\lambda,x) \xrightarrow{\text{LDSE}} \mathrm{MSE}(A,\lambda^*,x)} \xrightarrow{\text{PME}} \mathrm{MSE}(A^*,\lambda^*,x)}$$

Fig. 5. Optimization process of modified PME.

to switch on/off the coefficient optimization process described in Section 5.

Our test problems are partitioned into two groups: Exact fitting problems (Section 6.1) and test problems from literature (Section 6.2). As we know, practical engineering applications of symbolic regression are generally complex (see Section 4.1), so we do not expect a GP algorithm (including PME) capable of getting the exact solution. However, exact fitting problems (they are usually man-made) can help us evaluate PME's capability of 'structure optimization'. Test problems from literature give a comprehensive test of PME. They help us evaluate PME's overall capability of 'structure and coefficient optimization'.

In all our numerical experiments (except Prob. 6 in Section 6.2), uniform inner grid points are used as training points. For example, if we consider a function within the region $[0,1]\times[-1,0]$ and we want to use two training points at each direction, there will be four training points located at (0.333, -0.333), (0.333, -0.666), (0.666, -0.333) and (0.666, -0.666). Note that the number of inner grid points will be quite large for high dimensional problems. To avoid unnecessary sampling, experimental design method such as Latin Hypercube (McKay et al., 1979) and Uniform Design (Fang and Wang, 1994) might be applied. However, according to our experience, too few training points might result in unreliable results.

In our tests, the current and intermediate expressions are initialized as follows:

$$f=1,\quad f_1=1\quad \text{and } f_2=1.$$

6.1. Exact fitting problems

In this test group, 15 problems (see Table 5) are chosen to test PME's capability of 'structure optimization'. The coefficient optimization process of PME is switched off. All of them have exact fitting models in the search space. So you may expect to get the target model itself by using PME. However, PME might give you an equivalent alternative to the target expression. Suppose you have a set of training points whose target model is $\ln(x_1+x_2)+\sin(x_1+x_2)$, then PME might give you the equivalent solution $\sin(x_2+x_1)+\ln(x_2+x_1)$, $\sin(x_1+x_2)+\ln(x_1+x_2)-\ln(1)$, or $\ln((\exp(\sin(x_1+x_2)))*(x_1+x_2))$, etc. Therefore, we suggest the users using symbolic processors such as Maple or Mathematica to simplify the result from PME.

In case the target model is out of the search space, PME is capable of providing an approximate model. Suppose that we use Table 2 as the mapping rules, and the target model is $\sqrt{x_1+x_2}$, then it is not in the search space. (Note that the coefficient optimization process of PME is switched off.) By PME, we can get its best approximate model within the search space $\ln(x_1+x_2)/\cos(1)$.

The control parameters of PME are set as follows. The height of the parse-matrix $m=6$. The population size $N=100$ and the maximum generations for re-initialization T is set to 10,000 (i.e., $t\le T$). The tournament size in steady state GA $s=5$. PME will terminate immediately if the exact target function (or its equivalent alternative) is detected, and restart automatically if it fails until generation T. To reduce the influence of randomness, 100 runs for each test problems are carried out. The average number of residual evaluations and the average number of restarts are recorded to show the efficiency of PME.

Table 5
Test models and performance of PME.

No.	Dim	Target model	Domain	No. samples	Ave no. eval	Ave no. start
1	1	$x^2-\sin x$	1,3]	8	3230	4.8
2	1	$\sin x+2x$	[1,3]	8	2071	1.1
3	1	$\cos x^2-x$	[1,3]	8	2603	2.3
4	1	$\sin x+\cos x-x$	[1,3]	8	3646	7.4
5	2	x_1+2x_2	$[1,3]^2$	16	1047	1.0
6	2	$(x_1+x_2)/x_2$	$[1,3]^2$	16	1701	1.3
7	2	$\sin(x_1+x_2)$	$[1,3]^2$	16	1537	1.0
8	2	$\cos(x_1+x_2)-x_2$	$[1,3]^2$	16	2798	1.5
9	2	$\ln(x_1+x_2)$	$[1,3]^2$	16	1584	1.2
10	2	$x_1-e^{x_1+x_2}$	$[1,3]^2$	16	3593	1.8
11	2	$\ln(x_1+x_2)+\sin(x_1+x_2)$	$[1,3]^2$	16	4911	5.7
12	2	$\sin(x_1^2-x_2)+\ln(x_1+x_2)$	$[1,3]^2$	16	7637	13.2
13	3	$x_1+x_2-x_3$	$[1,3]^3$	64	4505	1.9
14	3	$x_1x_2-x_3$	$[1,3]^3$	64	3164	2.7
15	3	$\sin(x_1x_2)+x_3$	$[1,3]^3$	64	3302	11.6

Table 5 shows the test models and PME's average performance of the 100 independent runs with different initial populations. The full names of the notations in Table 5 are the dimension of modelled system (Dim), the target model (Target model), the domain of the target model (Domain), the number of training points (No. samples), the average number of residual evaluations of all successful runs (Ave no. eval), and the average number of restarts of 100 runs to get the target model (Ave no. start).

The computation results from Table 5 show that PME can recover the target models for all these test problems. The computational cost depends on the complexity of target model.

6.2. Test problems from literature

In this test group, we take all test problems from Keijzer (2003) into account to give PME an overall evaluation of its performance, including its 'structure optimization' and 'coefficient optimization' capabilities. The coefficient optimization process of PME is switched on. The control parameters are set as follows. The population size of PME $N=500$, and the maximum generations for re-initialization T is set to 20,000, the maximum number of restart $M=20$. The tournament size in steady state GA $s=5$. In the embedded LDSE, the population size $N_{\mathrm{LDSE}}=20$, the maximum generations for termination $T_{\mathrm{LDSE}}=200$, the adsorption probability $p_a=0.1$ (Luo et al., 2012). The coefficients are bonded by $\lambda\in[-50,50]^6$. The default target accuracy $\varepsilon_{\mathrm{target}}=10^{-8}$.

Similar to the first test group, 100 runs for each test problems are carried out. The algorithm will exit immediately if the mean square error is small enough ($\mathrm{MSE}\le\varepsilon_{\mathrm{target}}$), and the number of restarts (no.start) is recorded.

Table 6 shows the overall performance of the modified PME in the 100 independent runs with different initial populations. We can see that PME can solve most of the problems (12 of 15) without changing any control parameters. The number of restart range from 1 to 9 (except Prob. 4), and the average number of restart to solve these problems is acceptable. By increasing the model complexity (set $m=7$), Prob. 15 is also solved. You can also expect PME capable of solving Prob. 4 by increasing m. But we would rather let PME concentrate on 'structure optimization' as discussed in Section 5. For detailed results, refer to Appendix A.

Table 6
Performance of modified PME on Keijzer'problems.

No.	No. samples	MSE	No. start range	Ave no. start	Remarks
1	100	$\leq \varepsilon_{\text{target}}$	[1, 4]	2.76	Solution is almost exact
2	100	$\leq \varepsilon_{\text{target}}$	[1, 4]	2.55	Solution is almost exact
3	100	$\leq \varepsilon_{\text{target}}$	[1, 4]	2.6	Solution is almost exact
4	100	$[1.556, 1.562] \times 10^{-2}$	[20, 20]	20	Failed to reach the target accuracy
5	1000	$\leq \varepsilon_{\text{target}}$	[1, 5]	3.35	Solution is almost exact
6	50	$[7.426, 7.937] \times 10^{-7}$	[1, 4]	2.26	($\varepsilon_{\text{target}} = 10^{-6}$, see Appendix A)
7	100	$\leq \varepsilon_{\text{target}}$	[1, 2]	1.12	Solution is almost exact
8	100	$\leq \varepsilon_{\text{target}}$	[1, 4]	2.92	Solution is almost exact
9	100	$\leq \varepsilon_{\text{target}}$	[1, 6]	4.26	Solution is almost exact
10	100	$\leq \varepsilon_{\text{target}}$	[1, 3]	1.47	Solution is almost exact
11	100	$\leq \varepsilon_{\text{target}}$	[1, 7]	5.68	Solution is almost exact
12	100	$\leq \varepsilon_{\text{target}}$	[1, 6]	4.03	Solution is almost exact
13	100	$\leq \varepsilon_{\text{target}}$	[1, 4]	2.94	Solution is almost exact
14	100	$\leq \varepsilon_{\text{target}}$	[1, 4]	3.51	Solution is almost exact
15	100	$\leq \varepsilon^{*}_{\text{target}}$	[2, 9]	6.97	(m=7, see Appendix A)

7. Conclusion

We have introduced a new symbolic regression method: parse-matrix evolution (PME). It uses parse-matrix as its chromosome presentation. It is easier than ever. It can easily be implemented in any programming language and free to control. Furthermore, it does not need any additional function parsing process. Numerical results show that PME can recover the structure of a target model with plenty of input-response data, provided that the structure is not so complex. PME is also capable of optimizing the structure and coefficients simultaneously.

PME is NOT just an alternative to linear/nonlinear regression. The true value of PME is its 'structure optimization' ability, which can be used to detect the underlying physical relationships between different parameters.

Further research is necessary to make PME more effective for symbolic regression. For example, more efficient search engine is desired. The steady-stage GA is used as the search engine of PME in this paper, but its efficiency is not satisfactory for complex problems with high level optimal subtrees. Other search engines might be more efficient.

The idea of PME is generic. Although PME is only applied to symbolic regression in this paper, it could be easily adapted for other automatic programming problems such as the analytical solution of partial differential equations (Tsoulos and Lagaris, 2006), credit classification (Brabazon and O'Neill, 2006), Santa Fe ant trail and symbolic integration, etc.

Acknowledgements

This work was partially supported by the National Natural Science Foundation of China (Grant No. 90916028).

Appendix A. Test results in detail

To comply with the literature, we use x, y, z instead of x_1,x_2,x_3 (which are used in our program). That is, $x=x_1, y=x_2, z=x_3$.

Prob 1–3. $f(x)=0.3x\sin(2\pi x)$, where $x\in[-1,1]$, $x\in[-2,2]$, $x\in[-3,3]$ respectively.
An example best individual output from PME is

$$A^{*}=\begin{pmatrix} 2 & 0 & 1 & 1\\ 3 & -2 & 1 & 0\\ 2 & 1 & -3 & 1\\ 0 & -3 & -1 & 1\\ 2 & -2 & 0 & 0\\ 0 & -2 & 2 & 1 \end{pmatrix}$$

and its corresponding analytical expression is $f=(\lambda_2*((x_1)*(\sin(\lambda_1*((1)*(x_1))))))*(1)=\lambda_2 x_1\sin(\lambda_1 x_1)$, where $\lambda_1=6.283$, $\lambda_2=0.300$. The coefficient vector optimized by LDSE is $\lambda^{*}=(6.283, 0.300, -2.570, 25.04, -38.57, 7.433)$.

Prob 4. $f(x)=x^3\exp^{-x}\cos(x)\sin(x)(\sin^2(x)\cos(x)-1)$, where $x\in[0,10]$.
Using the given control parameters, PME failed to get the exact solution or the approximate model with MSE $\leq 10^{-8}$ in all 100 runs.

Prob 5. $f(x,y,z)=30xz/(x-10)y^2$, where $x,z\in(-1,1)$, and $y\in(1,2)$.

An example best individual output from PME is

$$A^* = \begin{pmatrix} 2 & 0 & 1 & 1 \\ -1 & -2 & 0 & 0 \\ 2 & 2 & 2 & -1 \\ 2 & -1 & -3 & 0 \\ -2 & 1 & -1 & 1 \\ 2 & 3 & -2 & 0 \end{pmatrix}$$

and its corresponding analytical expression is $f=(x_3)*(\lambda_2*((x_1)/((\lambda_1*((1)*(x_1))-(1))*((x_2)*(x_2)))))=\lambda_2 xz/(\lambda_1 x-1)y^2$, where $\lambda_1=0.100$, $\lambda_2=3.000$. The coefficient vector optimized by LDSE is $\lambda^*=(0.100,3.000,24.61,-6.975,-8.670,-11.67)$.

Prob 6. $f(n)=\sum_{i=1}^{n} 1/n$, where $n\in\{1,2,\ldots,50\}$.
Using the given control parameters, PME failed to get the exact solution or the approximate model with $\text{MSE}\le 10^{-8}$ in all 100 runs. Only the approximate models with $\text{MSE}\le 10^{-7}$ are evolved. An example best individual (with $\text{MSE}=7.426\cdot 10^{-7}$) output from PME is

$$A^* = \begin{pmatrix} 2 & -3 & 0 & 1 \\ 1 & 1 & -2 & 0 \\ -4 & -3 & 1 & 0 \\ 4 & -3 & 1 & -1 \\ 2 & 0 & 0 & 1 \\ 1 & -2 & -1 & 0 \end{pmatrix}$$

and its corresponding analytical expression is $f=\lambda_2*((1)*(1))+\ln(x_1+\lambda_1*((1.0)*(1)))=\lambda_2+\ln(x+\lambda_1)$, where $\lambda_1=0.522$, $\lambda_2=0.576$. The coefficient vector optimized by LDSE is $\lambda^*=(0.522,0.576,30.74,-8.339,-9.744,-30.12)$.

Prob 7. $f(x)=\ln x$, where $x\in[1,100]$.
An example best individual output from PME is

$$A^* = \begin{pmatrix} 1 & -2 & -3 & 1 \\ 3 & 0 & -1 & 1 \\ 4 & -2 & -1 & -1 \\ 0 & -3 & 1 & -1 \\ -4 & -2 & 0 & 1 \\ -4 & 1 & -2 & 1 \end{pmatrix}$$

and its corresponding analytical expression is $f=\ln(x_1)=\ln x$.

Prob 8. $f(x)=\sqrt{x}$, where $x\in(0,100)$.
An example best individual output from PME is

$$A^* = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 4 & -2 & 1 & 0 \\ -4 & 1 & -1 & 1 \\ 0 & -3 & 1 & 0 \\ 4 & -2 & 0 & -1 \\ 2 & -3 & 0 & 1 \end{pmatrix}$$

and its corresponding analytical expression is $f=(\exp(\lambda_2*(\ln(x_1))))*(1)=\exp^{\lambda_2*\ln x}$, where $\lambda_2=0.500$. The coefficient vector optimized by LDSE is $\lambda^*=(15.28,0.500,-18.62,\ -45.25,\ -28.12,\ 29.97)$.

Prob 9. $f(x)=\text{arcsinh}(x)$, where $x\in(0,100)$.
An example best individual output from PME is

$$A^* = \begin{pmatrix} 2 & 1 & 1 & 1 \\ 1 & 0 & -3 & 0 \\ -4 & -3 & 1 & 1 \\ 4 & -2 & 1 & -1 \\ 1 & -3 & 1 & -1 \\ -4 & -3 & -2 & 0 \end{pmatrix}$$

and its corresponding analytical expression is $f=\ln(\exp(\lambda_2{*}(\ln(1+(x_1){*}(x_1))))+x_1)=\ln(\exp^{\lambda_2{*}\ln(1+x^2)}+x)$, where $\lambda_2=0.500$. The coefficient vector optimized by LDSE is $\lambda^*=(30.27,0.500,33.99,25.88,3.699,-44.93)$.

Prob 10. $f(x,y)=x^y$, where $x\in(0,1)$, and $y\in(0,1)$.
An example best individual output from PME is

$$A^*=\begin{pmatrix}-4 & 1 & 2 & 0\\ -1 & -1 & 1 & 1\\ 2 & 1 & 0 & -1\\ 2 & -1 & 2 & 0\\ 3 & 0 & 1 & -1\\ 4 & -1 & 1 & 0\end{pmatrix}$$

and its corresponding analytical expression is $f=\exp((\ln(x_1)){*}(x_2))=\exp^{y\ \ln x}$.

Prob 11. $f(x,y)=xy+\sin((x-1)(y-1))$, where $x\in[-3,3]$, and $y\in[-3,3]$.
An example best individual output from PME is

$$A^*=\begin{pmatrix}1 & 2 & 1 & 0\\ -1 & 0 & -3 & 1\\ 2 & 1 & 2 & 0\\ 1 & -2 & -3 & 1\\ 3 & -3 & 1 & -1\\ 1 & -1 & -3 & 1\end{pmatrix}$$

and its corresponding analytical expression is $f=(x_1){*}(x_2)+\sin(\lambda_1{*}(1-(x_2+x_1))+(x_1){*}(x_2))=xy+\sin(\lambda_1{*}(1-(x+y))+xy)$, where $\lambda_1=1.000$. The coefficient vector optimized by LDSE is $\lambda^*=(1.000,-6.116,35.55,-14.64,27.85,24.43)$.

Prob 12. $f(x,y)=x^4-x^3+y^2/2-y$, where $x\in[-3,3]$, and $y\in[-3,3]$.
An example best individual output from PME is

$$A^*=\begin{pmatrix}2 & 2 & 2 & 1\\ 2 & 1 & 1 & 0\\ -1 & -1 & 1 & -1\\ 2 & -3 & -1 & 0\\ -1 & -2 & 2 & 1\\ 1 & -3 & -1 & 0\end{pmatrix}$$

and its corresponding analytical expression is $f=\lambda_1{*}((x_2){*}(x_2))-(x_2)+((x_1){*}(x_1)-(x_1)){*}((x_1){*}(x_1))=\lambda_1 y^2-y+x^2(x^2-x)$, where $\lambda_2=0.500$. The coefficient vector optimized by LDSE is $\lambda^*=(0.500,41.49,-2.118,39.83,11.00,16.06)$.

Prob 13. $f(x,y)=6\sin(x)\cos(y)$, where $x\in[-3,3]$, and $y\in[-3,3]$.
An example best individual output from PME is

$$A^*=\begin{pmatrix}-2 & 1 & -2 & -1\\ 3 & -1 & 1 & 0\\ 2 & 2 & 1 & -1\\ -3 & 2 & -2 & 1\\ 3 & 1 & -3 & 0\\ 2 & -1 & -2 & 0\end{pmatrix}$$

and its corresponding analytical expression is $f=(\sin(x_1)){*}(\lambda_1{*}(\cos(x_2)))=\lambda_1\sin x\cos y$, where $\lambda_1=6.000$. The coefficient vector optimized by LDSE is $\lambda^*=(6.000,-21.19,-24.50,-13.65,-13.70,39.91)$.

Prob 14. $f(x,y)=8/(2-x^2+y^2)$, where $x\in[-3,3]$, and $y\in[-3,3]$.
An example best individual output from PME is

$$A^*=\begin{pmatrix}2 & 2 & 2 & 0\\ 2 & 1 & 1 & -1\\ 1 & -3 & -1 & 1\\ 1 & -2 & 0 & 1\\ -2 & 0 & -2 & 0\\ 0 & 1 & 1 & 1\end{pmatrix}$$

and its corresponding analytical expression is $f=(1)/(\lambda_2{*}(\lambda_1{*}((x_1){*}(x_1)+(x_2){*}(x_2))+1))=1/\lambda_2\cdot(1+\lambda_1\cdot(x^2+y^2)))$, where $\lambda_1=0.500$, and $\lambda_2=0.250$. The coefficient vector optimized by LDSE is $\lambda^*=(0.500,0.250,33.61,-19.24,44.85,-27.30)$.

Prob 15. $f(x,y)=x^3/5+y^3/2-y-x$, where $x\in[-3,3]$, and $y\in[-3,3]$.

Using the given control parameters, PME failed to get the exact solution or the approximate model with the default accuracy (MSE $\le 10^{-8}$) in all 100 runs. However, if we reset the height of parse matrix $m=7$ or greater, PME can get the desired target function. An example best individual output from PME is

$$A^*=\begin{pmatrix}2 & 2 & 2 & 1\\ 2 & 2 & -2 & 0\\ -1 & -2 & 1 & 0\\ 2 & 1 & 1 & 1\\ 2 & -2 & 1 & -1\\ -1 & -3 & 2 & -1\\ 1 & -3 & -1 & 1\end{pmatrix}$$

and its corresponding analytical expression is $f=(\lambda_2{*}((x_1){*}(x_1))){*}(x_1)-(x_2)+(x_2){*}(\lambda_1{*}((x_2){*}(x_2)))-(x_1)=\lambda_2x^3-y+\lambda_1y^3-x$, where $\lambda_1=0.500$, and $\lambda_2=0.200$. The coefficient vector optimized by LDSE is $\lambda^*=(0.500,0.200,42.93,6.291,-10.49,26.44,-33.48)$.

References

Blickle, T., Thiele, L., 1996. A comparison of selection schemes used in evolutionary algorithms. Evolut. Comput. 4 (4), 361–394.

Brabazon, A., O'Neill, M., 2006. Credit classification using grammatical evolution. Informatica 30, 325–335.

Brameier, M., Banzhaf, W., 2007. Linear Genetic Programming. Springer, New York.

Chatterjee, S., Hadi, A.S., 2006. Regression Analysis by Example, 4th edition John Wiley and Sons.

Fang, K.T., Wang, Y., 1994. Number-Theoretic Methods in Statistics. Chapman & Hall, London.

Goldberg, B., 1996. Functional programming languages. ACM Comput. Surv. 28 (1), 249–251.

Hartl, D.L., Jones, E.W., 2008. Genetics: Analysis of Genes and Genomes, 7th edition Jones and Bartlett Publishers.

Keijzer, M., 2003. Improving symbolic regression with interval arithmetic and linear scaling In: Proceedings of the 6th European Conference on Genetic Programming. Essex, UK, vol. 2610, pp. 70–82.

Koza, J.R., 1992. Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA.

Luo, C., Yu, B., 2012. Low dimensional simplex evolution: a hybrid heuristic for global optimization. J. Glob. Optim. 52 (1), 45–55.

Luo, C., Zhang, S.-L., Yu, B., 2012. Some modifications of low-dimensional simplex evolution and their convergence. Optimization Methods and Software, Online first. http://dx.doi.org/10.1080/10556788.2011.584876.

Mabu, S., Hirasawa, K., Hu, J., 2007. A graph-based evolutionary algorithm: genetic network programming (GNP) and its extension using reinforcement learning. Evol. Comput. 15 (3), 369–398.

Martí, R., 2003. Multi-start methods. In: Glover, F., Kochenberber, G.A. (Eds.), Handbook of Meta Heuristics. Kluwer Academic Publishers, pp. 355–368.

McKay, M., Beckman, R., Conover, W., 1979. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. Technometrics 21, 239–246.

McKay, R., Hoai, N., Whigham, P., Shan, Y., O'Neill, M., 2010. Grammar-based genetic programming: a survey. Genet. Program. Evol. Mach. 11 (3), 365–396.

O'Neill, M., Ryan, C., 2001. Grammatical evolution. IEEE Trans. Evol. Comput. 5, 349–358.

O'Neill, M., Brabazon, A., 2006. Grammatical swarm: the generation of programs by social programming. Nat. Comput. 5 (4), 443–462.

Patelli, A., Ferariu, L., 2010. Elite based multiobjective genetic programming in nonlinear systems identification. Adv. Electr. Comput. Eng. 10 (1), 94–99.

Poli, R., 1996. Discovery of Symbolic, Neuro-Symbolic and Neural Networks with Parallel Distributed Genetic Programming. Technical Report CSRP-96-14, University of Birmingham, School of Computer Science, pp. 1–14.

Poli, R., Langdon, W., Mcphee, N., 2008. A Field Guide to Genetic Programming. Lulu Enterprises UK Ltd. (With contributions by J. R. Koza).

Searson, D.P., Leahy, D.E., Willis, M.J., 2010. GPTIPS: an open source genetic programming toolbox for multigene symbolic regression In: Proceedings of the International Multiconference Engineers and Computer Scientists 2010, Hong Kong, vol. 1, pp. 77–80.

Streeter, M., Becker, L.A., 2003. Automated discovery of numerical approximation formulae via genetic programming. Genet. Program. Evol. Mach. 4 (3), 255–286.

Tsoulos, I.G., Lagaris, I.E., 2006. Solving differential equations with genetic programming. Genet. Program. Evol. Mach. 7 (1), 33–54.

Vladislavleva, E., Smits, G., den Hertog, D., 2009. Order of nonlinearity as a complexity measure for models generated by symbolic regression via Pareto genetic programming. IEEE Trans. Evol. Comput. 13 (2), 333–349.

Vladislavleva, E., Smits, G., den Hertog, D., 2010. On the importance of data balancing for symbolic regression. IEEE Trans. Evol. Comput. 14 (2), 252–277.

Yang, Y.W., Wang, C., Soh, C.K., 2005. Force identification of dynamic systems using genetic programming. Int. J. Numer. Methods Eng. 63 (9), 131–1288.

Yao, L., Lin, C.-C., 2009. Identification of nonlinear systems by the genetic programming-based Volterra filter. IET Signal Process. 3 (2), 93–105.