Hindawi Publishing Corporation
Computational Intelligence and Neuroscience
Volume 2016, Article ID 1021378, 17 pages
http://dx.doi.org/10.1155/2016/1021378

Research Article

Using Genetic Programming with Prior Formula Knowledge to Solve Symbolic Regression Problem

Qiang Lu, Jun Ren, and Zhiguang Wang

Department of Computer Science and Technology, China University of Petroleum, Beijing 102249, China

Correspondence should be addressed to Qiang Lu; luqiang@cup.edu.cn

Received 28 May 2015; Revised 24 September 2015; Accepted 5 October 2015

Academic Editor: Christian W. Dawson

A researcher can infer mathematical expressions of functions quickly by using his professional knowledge (called Prior Knowledge). But the results he finds may be biased and restricted to his research field due to limitation of his knowledge. In contrast, Genetic Programming method can discover fitted mathematical expressions from the huge search space through running evolutionary algorithms. And its results can be generalized to accommodate different fields of knowledge. However, since *GP* has to search a huge space, its speed of finding the results is rather slow. Therefore, in this paper, a framework of connection between Prior Formula Knowledge and *GP* (*PFK-GP*) is proposed to reduce the space of *GP* searching. The PFK is built based on the Deep Belief Network (*DBN*) which can identify candidate formulas that are consistent with the features of experimental data. By using these candidate formulas as the seed of a randomly generated population, *PFK-GP* finds the right formulas quickly by exploring the search space of data features. We have compared *PFK-GP* with Pareto *GP* on regression of eight benchmark problems. The experimental results confirm that the *PFK-GP* can reduce the search space and obtain the significant improvement in the quality of SR.

1. Introduction

Symbolic regression (SR) is used to discover mathematical expressions of functions that can fit the given data based on the rules of accuracy, simplicity, and generalization. As distinct from linear or nonlinear regression that efficiently optimizes the parameters in the prespecified model, SR tries to seek appropriate models and their parameters simultaneously for a purpose of getting better insights into the dataset. Without any prior knowledge of physics, kinematics, and geometry, some natural laws described by mathematical expressions, such as Hamiltonians, Lagrangians, and other laws of geometric and momentum conservation, can be distilled from experimental data by the Genetic Programming (*GP*) method on SR [1].

Since SR is an NP-hard problem, some evolutionary algorithms were proposed to find approximate solutions to the problem, such as Genetic Programming (*GP*) [2], Gene Expression Programming (GEP) [3], Grammatical Evolution (GE) [4, 5], Analytic Programming (AP) [6], and Fast Evolutionary Programming (FEP) [7]. Moreover, recent researches in SR problem have taken into account machine learning (ML) algorithms [8–10]. All of the above algorithms randomly generate candidate population. But none of them can use various features of known functions to construct mathematical expressions adapted for describing the features of given data. Therefore, these algorithms may exploit huge search space that consists of all possible combinations of functions and its parameters.

Nevertheless, a researcher always analyzes data, infers mathematical expressions, and obtains results according to his professional knowledge. After getting experimental data, he observes the data distribution and their features and analyzes them with his knowledge. Then, he tries to create some mathematical models based on natural laws. He can obtain the values of coefficients in these models through regression analysis methods or other mathematical methods. And he evaluates the formulas which are mathematical models with the values by using various fitness functions. If the researcher finds some of the formulas that fit the experimental data, he can transform and simplify these formulas and then obtain the final formula that can represent the data. Furthermore, his rich experience and knowledge can help him to reduce the searching space complexity so that he can find the best

fit mathematical expression rapidly. As the researchers use their knowledge to discover the best fitted formulas, the methods that inject domain knowledge into the process of SR problem solving have been proposed to improve performance and scalability in complex problem [11–13]. The domain knowledge, which is manually created by the researcher's intuition and experience, is of various formulas which are prior solutions to special problems. If the domain knowledge automatically generates some fitted formulas that are used in evolutionary search without the researcher involvement, the speed of solving SR problem will be quickened. A key challenge is how to build and utilize the domain knowledge just like the researcher does.

In this paper, we present a framework of connection between Prior Formula Knowledge and *GP* (*PFK-GP*) to address the challenge:

(i) We classify a researcher's domain knowledge into PFK Base (*PFKB*) and inference ability after analyzing the process that a research discovered formulas from experimental data (Section 2). *PFKB* contains two primary functions: classification and recognition. The aim of two functions is to generate feature functions which can represent the feature of experimental data.

(ii) In order to implement classification, we use the deep learning method *DBN* [14, 15] which, compared with other shallow learning methods (Section 3.1), can classify experimental data into a special mathematical model that is consistent with data features. However, the classification method may lead to overfitting because the method can only categorize experimental data into known formula models which come from the set of training formula models.

(iii) Therefore, recognition is used to overcome the overfitting. It can extract mathematical models of functions that can show full or partial features of experimental data. Three algorithms *GenerateFs*, *CountSamePartF*, and *CountSpecU* (see Algorithms 2, 3, and 4) are designed to implement recognition. For example, from the dataset generated by $f(x) = \exp(\sin(x) + x^3/(8 * 10^5))$, the basic functions sin, exp, and cube can be found by the above three algorithms. In Figure 1, the function sin shows the periodicity of data, and exp or cube shows the growth rate of data. Therefore, these basic functions (called feature functions) can describe some features of the dataset.

(iv) The inference ability is concluded to the searching ability of evolutionary algorithm. As researches infer mathematical models, *GP* is used to combine, transform, and verify these models. These feature functions that are generated by *PFKB* are selected to be combined into the candidate population in the light of algorithm *randomGenP* (see Algorithm 5). With the candidate population, *GP* can get convergent result quickly because it searches answers in a limit space which is composed of various feature functions. Through experiment on eight benchmark problems (Table 5 E_1–E_8), the results demonstrate that *PFK-GP*, compared with Pareto optimization *GP* [16, 17], shows the significant improvement in accuracy and convergence.

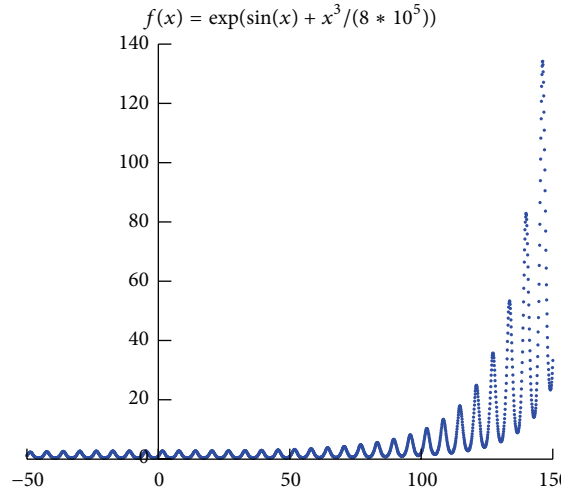

Figure 1: The function $f(x) = \exp(\sin(x) + x^3/(8 * 10^5))$.

2. Background

2.1. Definition and Representation of Mathematical Expression. In this section, we will define concepts about SR problem and show how to represent these concepts by applying *BNF* expression. For SR problem, the word "formula" is the general term which describes mathematical expression that fits the given data. We define a formula model is a special mathematical model in which formulas have the same relationships and variables except for different coefficient values. Relationships can be described by operators, such as algebraic operators, functions, and differential operators (http://en.wikipedia.org/wiki/Mathematical_model). Therefore, a formula model is a set where each element is a formula. For example, the two formulas $0.1 * \sin(x) + 0.7 * \log(x)$ and $0.3 * \sin(x) + 0.9 * \log(x)$ belong to the formula model $a_1 * \sin(x) + a_2 * \log(x)$. Data that are represented by different formulas in one formula model may have similar features which are data distributions, data relationships between different variables, data change laws, and so on, because these formulas have the same relationships.

In order to represent a formula model and its corresponding formulas, we define the following *BNF* expressions:

F:=C | S,

C:=S"("C")" | S,

S:=B"("AX, AX")" | U"("AX")",

B:=" + " | " − " | " ∗ " | "/",

U:="sqrt" | "log" | "tanh" | "sin" | "cos" | "exp" | "tan" | "abs" | "quart" | "cube" | "square" ⋯,

A:=a_1 | a_2 | a_3 | ⋯ | a_n,

X:=x_1 | x_2 | x_3 | ⋯ | x_m,

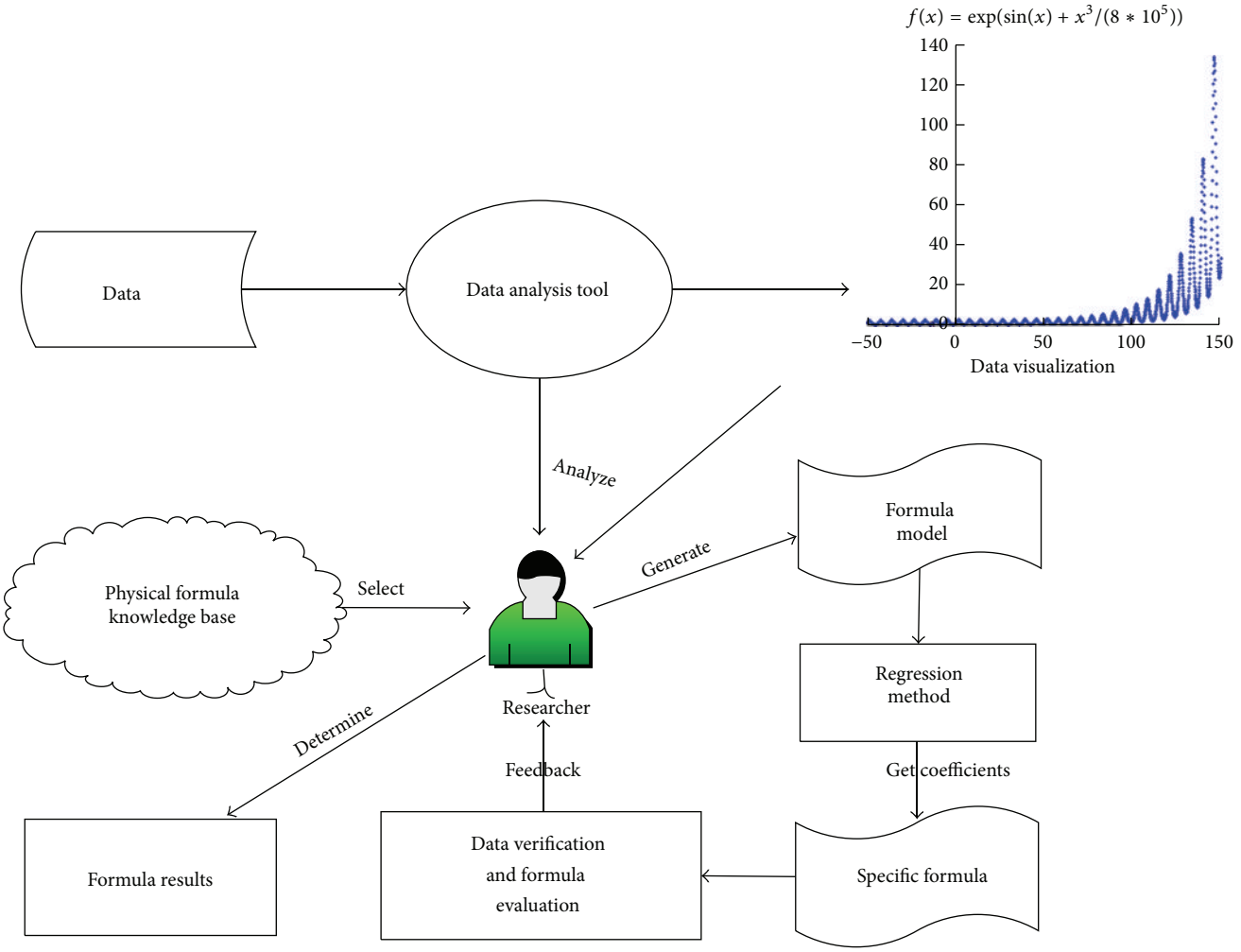

Figure 2: The process that researchers study the SR problem.

where F is a formula model. X is a parameter set. A indicates a coefficient set. B is a set of binary functions, while U is a set of unary functions. S is a set of atomic functions which does not contain any subfunctions. C is a set of complex functions which contains complex functions in C and atomic functions in S. With the above definitions, any formulas and its corresponding model can be shown by these *BNF* expressions. For instance, the formula $\exp(\sin(x) + x^3/(8 * 10^5))$ is represented by F and C, and its subfunction $\sin(x)$ is represented by U. The constants 8, 3, and 10^5 are shown by elements in A. With these *BNF* expressions, a formula model can be transformed into one tree. And the tree is a candidate individual in population of *GP* solving SR problem. Every subtree in the tree is a subformula which can show special data features. A subtree that shows features of experimental data is called feature-subtree. If a tree has more feature-subtrees, the tree is more likely to fit the data. How to construct the tree consisting of feature-subtrees is a key step in our method which is implemented by the algorithm *randomGenP* (see Algorithm 5).

2.2. The Process of Researcher Analyzing Data. The process that a researcher tries to solve SR problems is shown in Figure 2. He depends heavily on his experience which is obtained through a long-term accumulation of study and research. After a researcher collected experimental data, he discovers regular patterns from data by using the methods of data analysis and visualization. He then constructs formula models which were consistent with these regular patterns according to his experiences. After that, he computes the coefficient values in formula models by using appropriate regression methods and obtains some formulas from different formula models. According to results of evaluating these formulas, he chooses the formula that is most fitted to the data. If the formula cannot represent data features, he needs to reselect a new formula model and do the above steps until one fitting formula is found.

We think the researcher's experience and knowledge have two roles in processing SR problem. One role is Prior Formula Knowledge (PFK) which can help a researcher to quickly find fitted formulas that match experimental data features. Through study and work, the researcher accumulates his domain knowledge of various characteristics of formula model. When the researcher observes experimental data, he can apply his domain knowledge to recognize and classify the data. The other is the ability of inference and deduction which

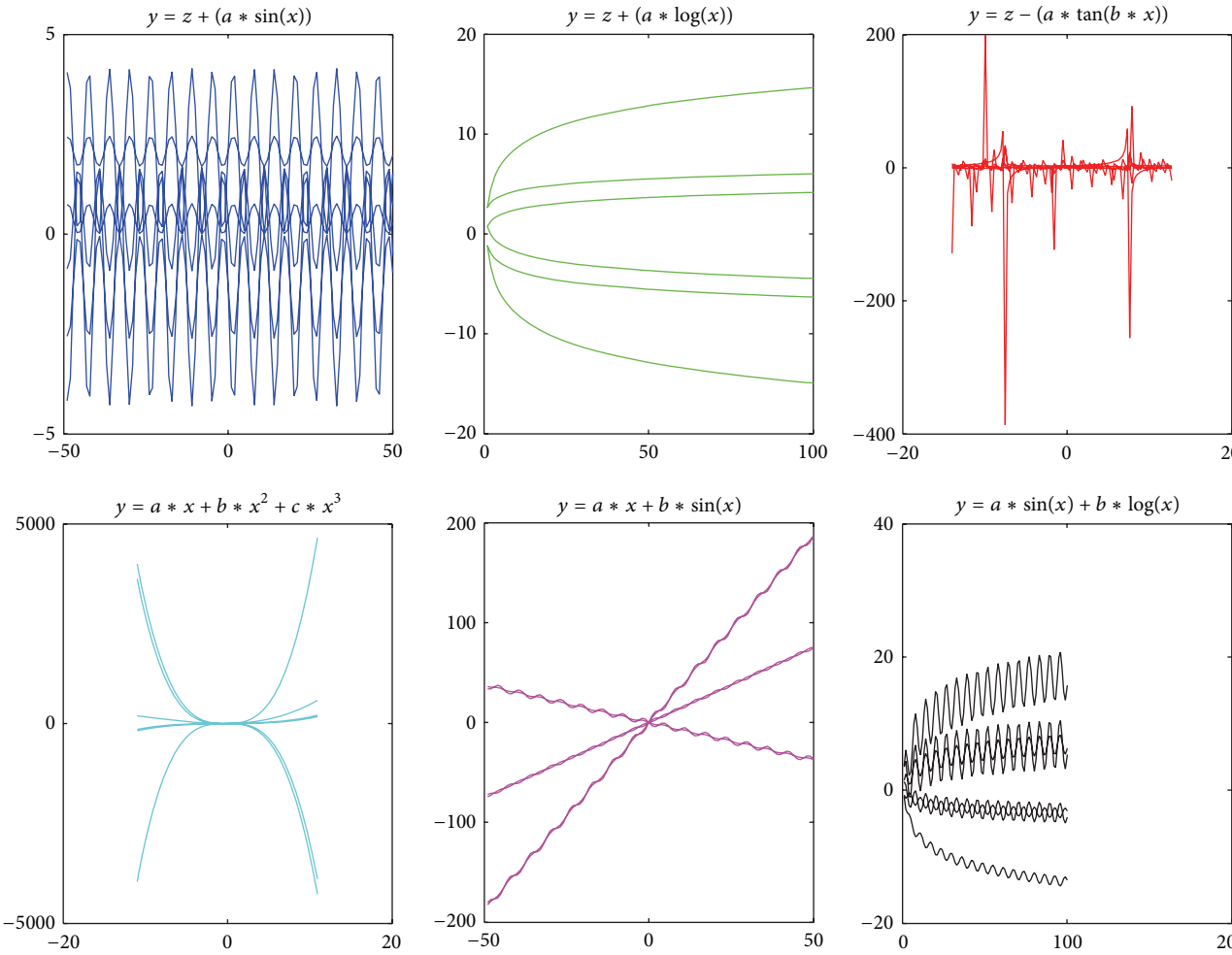

Figure 3: Curves are generated by six formula models with different coefficient value.

can help the researcher to combine, transform, and verify mathematical expression. We conclude that the PFK contains two primary functions: classification and recognition.

Classification. when experimental data features are in accord with characteristics of one formula model in PFK, the dataset can be categorized into the model. The prerequisite of classification is that different formula models have different characteristics in PFK Base. As shown in Figure 3, six families of curves are generated by six formula models taking different coefficient values. The curves in the same family show similar data features while the curves in different families show different data features. Therefore, we can infer that the curves (including surfaces and hypersurfaces) generated by different formula models can be classified according to their data features.

Although many machine learning algorithms such as linear regression [18], SVM [19], Boosting [20], and PCVMs [21] can be used to identify and classify data, it is difficult for these algorithms to classify these curves. That is because these algorithms depend on features that are extracted manually from data, while these features from different complex curves are difficult to be represented by a feature vector which is built based on the researcher's experiences. In contrast to these algorithms, DL can automatically extract features and have a good performance for the recognition of complex curves, such as image [15], speech [22], and natural language [23]. The *GenerateFs* algorithm (see Algorithm 2) based on *DBN* is shown to classify the data.

Recognition. Some formulas can represent remark features of curves generated by formula model. For example, after observing the curve in Figure 1, a researcher can easily infer that the formula sin or cos is one of formulas that constitute the curve because data in curve show periodicity. Therefore, these formulas are called feature functions that can be recognized or extracted by PFK. Algorithms *CountSamePartF* and *CountSpecU* (see Algorithms 3 and 4) are built to recognize the feature functions.

Recognition can help the researcher overcome overfitting of results that are generated by classification because classification can help researcher to only identify formula models from training set while recognition can help the researcher identify subformula models that are consistent with local data features.

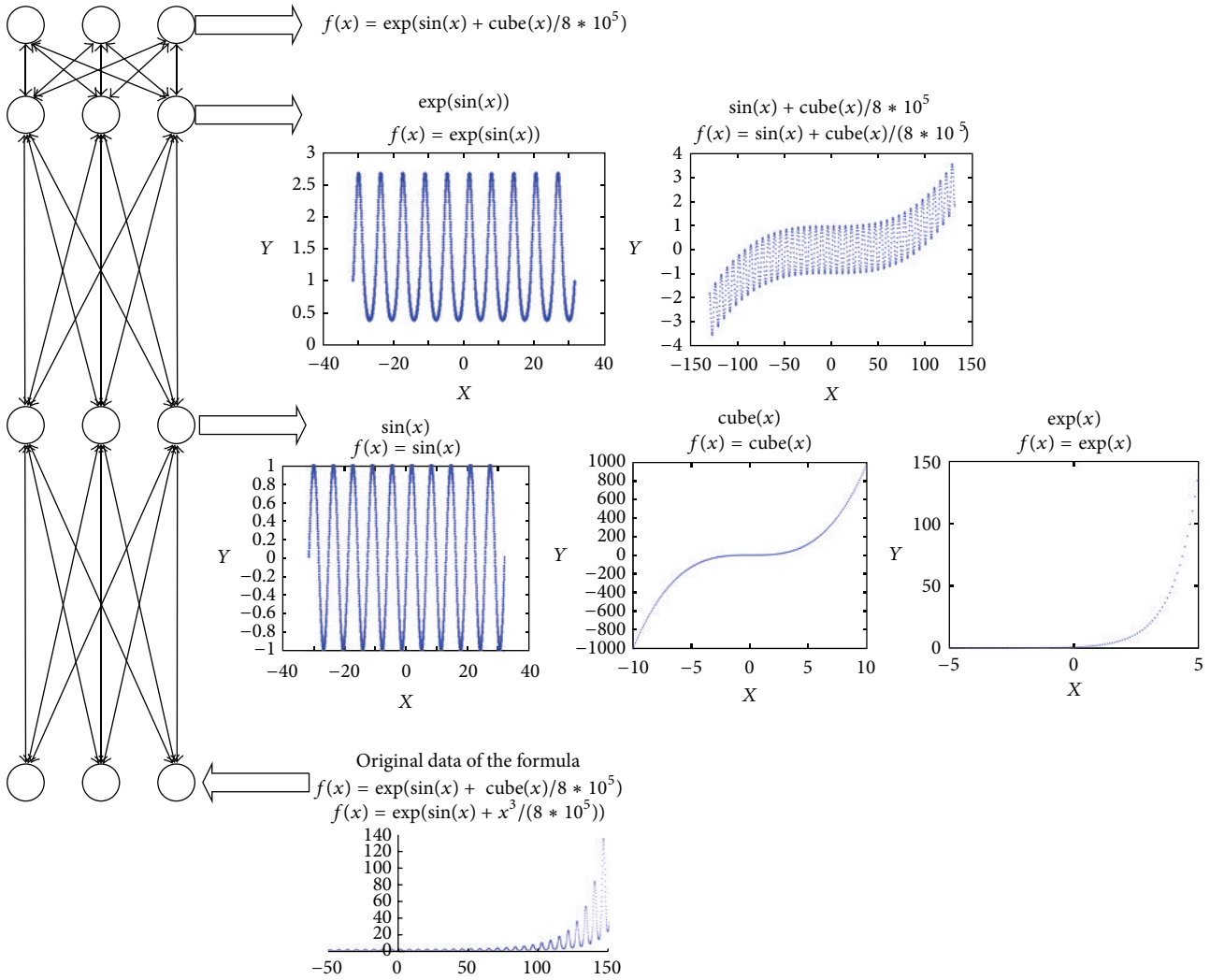

Figure 4: Illustration of the *DBN* framework.

The ability of inference and deduction is one of main measurements for evaluating performance of artificial intelligence methods. In the SR problem, *GP*, compared with other methods such as logical reasoning, statistical learning, and genetic algorithm, is a revolutionary method of searching fitting formulas because it can seek the appropriate formula models and their coefficient values simultaneously by evolving the population of formula individuals. Therefore, in the paper, we use *GP* as the method of inferring and deducing formulas.

To optimize *GP*, researchers have proposed various approaches, such as optimal parsimony pressure [24]; Pareto front optimization [17] and its age-fitness method [25] are used to control bloat and premature convergence in *GP*. In order to reduce the space complexity of searching formulas, the methods of arithmetic [26] and machine learning are injected into *GP*. In the paper, with the algorithm *randomGenP* (see Algorithm 5) about generating population and the method of Pareto front optimization, *PFK-GP* can research the formula model in the appropriate space and can find right formulas quickly.

3. Genetic Programming with Prior Formula Knowledge Base

3.1. Formula Prior Knowledge Base. The FPK needs to have the ability of identifying and classifying the formula model F based on data features. Although the features between formula models are different, it is difficult to extract features from data which are generated by these models because different formula models represent seemly similar but different features. Based on the above definitions in formula model, the features among functions in set S are different. The features between the function $s \in S$ and the function $c \in C$ may be similar if c is the parameter of s. As shown in Figure 4, these functions $\sin(x)$, $\text{cube}(x)$, and $\exp(x)$

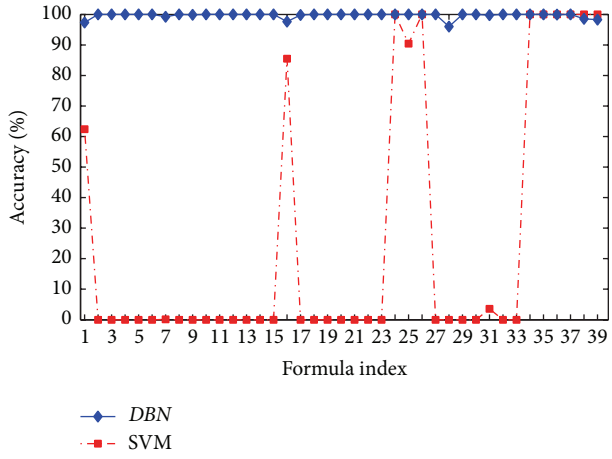

Figure 5: Accuracy result of SVM and *DBN* classifying P_1–P_{39}.

belonging to S constitute C, such as $\exp(\sin(x))$ and $\sin(x) + \text{cube}(x)/(8 * 10^5)$, and are shaped into the final function $\exp(\sin(x) + \text{cube}(x)/(8 * 10^5))$. $\sin(x)$ shows periodicity in results; $\log(x)$ shows slow variation trend in results; $\text{cube}(x)$ shows high variation trend in results. So, these features in the three functions are different. However, there are similar features between $\text{cube}(x)$ and $\sin(x) + \text{cube}(x)/(8 * 10^5)$, because the two functions have high variation trend in result. The shallow learning method such as SVM can hardly identify complex features of functions shown in Figure 5.

In this paper, *DBN* is used to classify data into a special formula model according to features that are automatically extracted from data. Generally, *DBN* is made up of multiple layers that are represented by Restricted Boltzmann Machine (RBM). As RBM is an universal approximate of discrete models [27], we can speculate that *DBN* which has many layers of RBM can recognize the features of data generated by complex function just like Convolutional Neural Network (CNN) classifying image [28]. The process of *DBN* recognizing formulas is illustrated in Figure 4. *DBN* can extract features from data, layer by layer. So, the lower RBM layer can represent the simple functions and the higher RBM layer can transform the simple functions into more complex functions.

We use the data generated by the different formula models F (see Table 7) as training samples. *DBN* is trained by these samples. The model that is finally gained by *DBN* training methods is *PFKB*, which is aimed at identifying the formula model that can represent features of the data. The process of *DBN* training is outlined in algorithm *TrainDBN* (Algorithm 1) which uses the same steps as mentioned in [14, 15].

PFKB is only changed with formula models. If there are no new trained formula models in an application, the algorithm *TrainDBN* will not be executed. When the number of trained formula models is large enough, little new formula model will appear, and *PFKB* will seldom be changed. In the paper, *TrainDBN* is performed exactly once in order to generate *PFKB*.

3.2. Classification and Recognition with PFKB. In order to deal with the problem of how to classify and recognize formula model from data, we should consider the problem from two aspects. One situation is that data can be represented by a special formula model from *PFKB*, while the other one is that data cannot be represented by a formula model from *PFKB*. In the first case, we exploit *PFKB* to identify formula models of data by *DBN* classification. Based on ordered results of *DBN* classification, we gain a set of formula models ($Fs = f_1, \ldots, f_s$) which are most similar to features of the data. The process that deals with the first case is outlined in algorithm *GenerateFs*. The algorithm is fast because *PFKB* has been built by *TrainDBN*, and s is small integer value.

In the second case, when a researcher observes laws that are hidden in experimental data, he often tries to find some formulas C which are consistent with partial features of the data. Therefore, we propose the two assumptions as follows.

Assumption 1. More formula models fs have the same subformula model pf in the set Fs which is the result of *GenerateFs* running, more strongly that the pf can express features of data.

In order to compute the same pf in Fs, we express the formula model as the string of expression and seek the same part of them by using intersection between the two strings (without considering the elements in sets X and A). Define the intersection between two expressions as follows:

$$f_i \cap f_j = c_1, \ldots, c_k,$$
$$c_m \cap c_n = \emptyset, \quad (1)$$
$$m \neq n,\ c_n \in C,\ c_n \notin S,\ 1 \leq m,\ n \leq k,\ 1 \leq i,\ j \leq k.$$

For example, $f_1 = z + a * \cos(x) + \tan(x)/(\exp(x) + \log(x))$, $f_2 = z + a * \cos(x) + \text{abs}(x)/(\exp(x) + \log(x))$, $f_1 \cap f_2 = \{z + a * \cos(x), \exp(x) + \log(x)\}$. The method, which obtains pf whose frequency of occurrence in f is larger than threshold t, is described as the algorithm *CountSamePartF*.

For verifying Assumption 1, we apply the dataset from E_0 (see Table 5) as the testing set and get the identifying results $Fs = \{P_{18}, P_{15}, P_{34}, P_{11}, P_3\}$ from Table 7 through the algorithm *GenerateFs*. The intersections between the top two formulas $P_{18} \cap P_{15}$ are $(\text{sqrt}(x_0))/(\tan(x_1))$ and $(\tan(x_0))/(\exp(x_1))$, which are partial mathematical expressions in E_0. And we use the dataset from Table 6 to test E_0 and get the identifying results $Fs = \{T_7, T_6, T_8, T_9, T_1\}$ through *GenerateFs* algorithm. The intersection between two expressions is as follows: $T_7 \cap T_6 = \{\text{sqrt}(x_0)/\tan(x_1)\}$, $T_8 \cap T_9 = \{\tan(x_0)/\exp(x_1), \cos(x_0) * \sin(x_1)\}$. We find that the elements which have more frequency of occurrence in the intersections set are more likely to express some data features. The above two experiments illustrate that Assumption 1 is rational.

Assumption 2. If function $u \in U$ exists in Fs obtained by *GenerateFs* and the number of the same u is larger than threshold t, we can conclude that u can show some local data features.

Input: $X1$, $Y1$, $X2$, $Y2$ ($X1$, $Y1$ are training data; $X2$, $Y2$ are testing data)
Output: *PFKB*
(1) *Initial*(*DL*, *opts*) // initial the structure of DL and the parameters opts
(2) *DL* = *DLsetup*(*DBN*, *X*1, *opts*) // layer-wise pre-training DL
(3) *DL* = *DLtrain*(*DBN, X*1, *opts*) // build up each layer of DL to train
(4) *nn* = *DLunfoldtonn*(*DBN*, *opts*) // after training each layer, passing the parameters to *nn*
(5) *PFKB* = *nntrain*(*nn, X*1, *Y*1, *opts*) // fine-tune the whole deep architecture
(6) *accuracy* = *nntest*(*PFKB, X*2, *Y*2) // *accuracy* is the criterion of the quality of *PFKM*, if it is too small, then re-training after adjusting the model architecture or parameters
(7) *return PFKB*

Algorithm 1: *TrainDBN*: training *DBN* to generate the *PFKB*.

Input: *X*, *PFKB*, *s* (*X* is the dataset; *s* is the number of formula models whose features can show the dataset)
Output: *Fs* (formula model vector which are used in generating the initial population of *GP*)
(1) *Ftemp* = *predictModel*(*PFKB*, *X*) // as the intermediate data, it is the original result that exploit the *PFKB* to predict without sorting by the fitness
(2) *Ftemp* = *sortModelByFit*(*Ftemp*) // sort the models in order of decreasing fitness
(3) *for i* = 1 : *s*
(4) *Fs*(*i*) = *Ftemp(i)*
(5) *end*
(6) *return Fs*

Algorithm 2: *GenerateFs*.

The function $b \in B$ except x^y is common function, which has a high probability of occurrence in mathematical expressions. Therefore, it is difficult to express special data features. Compared with B, the function $u \in U$ can show obvious features of data. For instance, $\sin(x)$ presents the periodicity of data and $\log(x)$ represents data features about extreme increase or decrease. The method, which obtains the special function u that can show the local data features, is outlined as the algorithm *CountSpecU*.

For verifying Assumption 2, we also choose the dataset which are generated from E_0 (see Table 5) as the testing data and apply the *CountSpecU* algorithm to calculate the special u among $Fs = \{P_{18}, P_{15}, P_{34}, P_{11}, P_3\}$. The result of the algorithm is shown in Table 1. We find the result $specU = \{\tan, \cos, \text{sqrt}, \exp, \sin\}$ (sin and cos are the operators of the same kind) is part of E_0. Hence, we can discover that the u set, which is gained by the algorithm *CountSpecU*, can show local features of the dataset.

Table 1: The result of U in Fs computed by algorithm *CountSpecU* (see Algorithm 4).

u in Fs	Frequency of occurrence
tan	3
cos	2
sqrt	1
exp	1
log	1

3.3. GP with Prior Formula Knowledge Base. In order to deal with SR problem, *GP* is executed to automatically composite and evolve mathematical expression. The process of *GP* is similar to the process that a researcher transforms formula models and obtains fitting formulas based on his knowledge. Since those algorithms in *PFKB*, which is created based on analyzing the process of how a research infers fitted formulas, can recognize formula models that are consistent with data features, we combine these formula models of *PFKB* recognizing into the process of *GP* in order to reduce the searching space and increase the speed of discovering right solutions.

When initializing *GP* algorithm, we can select candidate formulas from *Fs*, *C*, and *specU* as individuals in population of *GP*. The sets *Fs*, *C*, and *specU* are gained by the above algorithms in *PFKB*. Therefore, the *PFKB* is injected into the process of population generating. And this population can contribute to reserving data features as much as possible and reducing the searching space because these individuals commonly have good fitness value. With the population, *GP* algorithm can speed up the convergence and improve the accuracy of SR results. However, it may lead to the bias results. To overcome the problem, some random individuals must be imported into the population. The process of population creating is as follows.

Firstly, the elements in sets *Fs* and *C* are inserted into the population. Then, the set *specU* and the candidate function sets *B* and *U* are merged into the new candidate function queue *Q*. And the number of elements in *specU*

Input: t, Fs
Output: C (ordered local expressions set that are sorted according to frequency of occurrence which is larger than t)
(1) $C = F = \emptyset$
(2) *for each pair* $\langle f_i, f_j \rangle$ *in* Fs
(3) $F_{ij} = f_i \cap f_j$
(4) *for each* c_m *in* F_{ij}
(5) *if* $c_m \in F$
(6) *change F's element* c_m^v *to* c_m^{v+1} // v indicates the number of times that c_m appears
(7) *else*
(8) *add* c_m^1 *into* F
(9) *end*
(10) *end*
(11) *for each* c_m^v *in* F
(12) *if* $v \geq t$
(13) *add* c_m^v *into* C
(14) *end*
(15) *sort*(C)
(16) *return* C

Algorithm 3: *CountSamePartF*.

Input: t, Fs
Output: *specU* (ordered *specU* function set that are sorted according to frequency of occurrence which is larger than t)
(1) $U = F = \emptyset$
(2) *for each* f_i *in* Fs
(3) *for each* u_m *in* f_i
(4) *if* $u_m \in F$
(5) *change F's element* u_m^v *to* u_m^{v+1} // v indicates the number of times that u_m appears
(6) *else*
(7) *add* u_m^1 *into* F
(8) *end*
(9) *end*
(10) *for each* u_m^v *in* F
(11) *if* $v \geq t$
(12) *add* u_m^v *into specU*
(13) *end*
(14) *return specU*

Algorithm 4: *CountSpecU*.

Input: Fs, C, *specU*, B, U, n where, n represents the number of individuals which are generated, B and U is the candidate function library)
Output: P (population)
(1) $I = Fs \cup C$
(2) *for each i in I*
(3) *add i into P*
(4) *end*
(5) $k = |specU|$
(6) $Q = specU + B + U$ // add the elements of *specU*, B and U into the queue Q successively, $u_m^v \in U$, add u_m^v into Q
(7) $P\_temp = traditionalRandomIndividual(Q, k)$
(8) *add P_temp into P*
(9) $k = n - |I| - k$
(10) $P\_temp = traditionalRandomIndividual(B + U, k)$
(11) *add P_temp into P*
(12) *return P*

Algorithm 5: *randomGenP*.

Input: data, *PFKB*, t_1, t_2, B, U, n, k, g, *interval*
Output: F (candidate formulas set)
(1) $Fs = GenerateFs(data, PFKB)$
(2) $C = CountSamePartF(t_1, Fs)$
(3) $U = CountSpecU(t_2, Fs)$
(4) $P = randomGenP(Fs, C, specU, B + U, n)$
(5) *while* (*bestFitness* <= *threshold* && $i < g$)
(6) $P = crossover(P)$
(7) $P = mutate(P)$
(8) $Pt = ParetoOptimise(P)$ // prevent the formula model too complex
(9) $Pt\_fitnees = EvaluatePopulation(Pt)$
(10) *bestFitness*, $F = Selectbest(Pt, Pt\_fitness, k)$ // choose the best k individuals and get the best fitness value from the individuals
(11) *if* i *mod interval*
(12) $P_1 = randomGenP(F, C, specU, B + U, n/2)$
(13) $P_2 = traditionalRandomIndividual(B + U, n/2)$
(14) $P = P_1 \cup P_2$
(15) *else*
(16) $P_1 = traditionalRandomIndividual(B + U, n - k)$
(17) $P = P_1 \cup P$
(18) *end*
(19) i++
(20) *end*
(21) *return* F

Algorithm 6: *PFK-GP*.

is twice as much as the other elements in Q because $B \cup C \subseteq specU$. Those elements in *specU* are more likely to be part of individuals in the population after applying the method *traditionalRandomIndividual* [16] which is designed to generate randomly k individuals from the special function set. At last, the rest of individuals of population are created by *traditionalRandomIndividual* with sets B and U. The process of population generating is described as the algorithm *randomGenP*.

Generally, $|Fs| + |C| + |specU| < n/2$, where n is the number of individuals in population. Furthermore, in order to enhance the affection of *PFKB* in the process of *GP* evolution, the method *randomGenP* is used to create new individuals in every few generations of evolutionary computation. Meanwhile, the method of Pareto front [17] is introduced into the algorithm *PFK-GP* to balance the accuracy against the complexity of model. The detail of algorithm *PFK-GP* is shown in Algorithm 6.

4. Experiments

In the experiments, we employ *DBN* in the DeepLearnToolbox [30] to classify formula models and build the algorithm *PFK-GP* based on GPTIPS [29]. The 39 formula models in Table 7 are composed of formulas from [31, 32] and some formulas are created by ourselves. The data generated by these 39 formula models is used as training data of algorithm *DBN* to create *PFKB*. The formula models in Table 5 are used to generate the testing data for verifying accuracy of algorithms *GenerateFs* and *PFK-GP*. The formula models in Table 6 are devoted to validating the two algorithms *CountSamePartF* and *CountSpecU* (see Algorithms 3 and 4).

Table 2: The parameter values in algorithm of *DBN* and SVM.

DBN parameters	Value	SVM parameters	Value
The number of *DBN* layers	4	svm_type	c-svc
The size of *DBN* hide nodes	50	Kernel	Gaussian
The number of epochs	200	Gamma	0.07
Batch size	40	Coef	0
Momentum	0	Cost	1.0
Alpha	1	Degree	3.0
activation_function	sigm	Shrinking	1

For most formula models from Tables 5, 6, and 7, we sampled them by equal step taking their parameter values from the range [−49, 50]. For some particular formulas, we also sample them with a special equal step from special numerical scope. For example, the value x in sqrt(x) is in the range [0, 99], the value x in log(x) ranges between 1 and 100. We create 500 groups of different parameters value in each formula model. The coefficients in these formula models are fetched with equal step from the range [−2.996, 3.0]. When all coefficients of a formula model take special values, the formula model generates a formula, namely, a sample of the formula model. We create 7500 groups of different coefficients in each formula model. So, each formula model has 7500 samples where each sample has 500 groups of different parameters value. We take 6000 samples of these samples as training data and the others as test data.

Table 3: Parameter values in *GP* and *PFK-GP*.

Parameter	Value
Representation	GPTIPS [29] multigene syntax Number of genes: 1 Maximum tree depth: 5
Population size	50
Number of generations	1000
Selection	Lexicographic tournament selection
Tournament size	3
Crossover operator	Subtree crossover
Crosser probability	0.85
Mutation operator	Subtree mutation
Mutation probability	0.1
Reproduction probability	0.05
Fitness	$\frac{1}{N}\sqrt{\sum (y-\hat{y})^2}$
Elitism	Keep 1 best individual

Table 4: The best mathematical expression of *PFK-GP* finding.

Number	The best mathematical expression
E_1	$y = 0.7001 * \tan(x_1 * x_4 - 5.049) - 0.7001 * x_1 * \text{cube}(2.575) * \text{cube}(x_4) - 1.001$
E_2	$y = 7.214 * \sin(x_1) + 1.001 * \tan(x_2)$
E_3	$y = 0.25 * \text{square}(x_1 + x_2 - 6) - 0.2179$
E_4	$y = 1.001 - \frac{(1.332 * (4 * x_1 + \log(\text{square}(x_2))))}{(2 * \text{square}(x_2) + 5.585)}$
E_5	$y = x_2 - 2.092 \tanh(\text{square}(\sin(x_1))) + 0.8795$
E_6	$y = 6 * \cos(x_2) * \sin(x_1) - 0.00444$
E_7	$y = \sin(x_1) - 6 * x_1 + \text{square}(x_1) + 14$
E_8	$y = \log(x_2) + \text{sqrt}(x_1) + \sin(x_1) + 0.1823$

Table 5: Test data used in *PFK-GP*.

Number	Formula
E_0	$y = -1.97 + 1.25 * \frac{\text{sqrt}(x_0)}{\tan(x_1)} + \frac{\tan(x_0)}{\exp(x_1)} + \cos(x_0) * \sin(x_1)$
E_1	$y = \exp(2 * x_1 * \sin(pi * x_4)) + \sin(x_2 * x_3)$
E_2	$y = 3.56 + 7.23 * \sin(x_0) + \tan(x_3)$
E_3	$y = (x_0 - 3) * (x_3 - 3) + 2 * \sin((x_0 - 4) * (x_3 - 4))$
E_4	$y = \frac{(\text{quart}(x_1 - 3) + \text{cube}(x_2 - 3) - (x_2 - 3))}{(\text{quart}(x_2 - 4) + 10)}$
E_5	$y = \tanh(\cos(2 * x_0)) + x_3$
E_6	$y = 6 * \sin(x_0) * \cos(x_3)$
E_7	$y = \sin(x_0) + \text{square}(x_0) + 5$
E_8	$y = \text{sqrt}(x_0) + \log(1.2 * x_3) + \sin(x_0)$

We adopt *DBN* as the classification model and compare it with SVM that is implemented by the tool libsvm [33]. The training and testing data for the two algorithms are originated from formula models P_1–P_{39}. The parameter values in *DBN* and SVM are illustrated as Table 2. We take the first five formulas from *Fs* generated by *GenerateFs* as a result set of recognition. If the test formula is included in the set, we think that the recognition result is correct. The accuracy of recognition results of *DBN* and SVM is showed as Figure 5. The *DBN* method can help to classify all kinds of test data into its fitted formula models. However, the SVM method can only correctly classify several kinds of test data. The overall average accuracy of *DBN* classification is 99.65%, while the accuracy of SVM is 26.72%. The result demonstrates that *DBN* is more suitable for recognizing data generated by mathematical expression, because *DBN* can automatically extract features from the data, layer by layer, and is similar to composition of formula which is constituted by its subformulas.

We set parameters in *GP* with Pareto optimization (*PO-GP*) [29] and *PFK-GP* as shown in Table 3. For data generated by P_{13} (see Table 7, coefficient z is −2.098, a is −2.998), *PO-GP*

Table 6: Test data of two algorithms *CountSamePartF* and *CountSpecU* (see Algorithms 3 and 4).

Number	Formula
T_1	P_1
T_2	P_2
T_3	P_3
T_4	P_4
T_5	P_5
T_6	$y = z + a * \frac{\text{sqrt}(x_0)}{\tan(x_1)} + \sin(x_1)$
T_7	$y = z + a * \frac{\text{sqrt}(x_0)}{\tan(x_1)} + x_1$
T_8	$y = z + a * \frac{\tan(x_0)}{\exp(x_1)} + \cos(x_0) * \sin(x_1) + \sin(x_1)$
T_9	$y = z + a * \frac{\tan(x_0)}{\exp(x_1)} + \cos(x_0) * \sin(x_1) + x_1$
T_{10}	$y = z + a * \cos(x_0) * \sin(x_1) + \text{square}(x_1)$

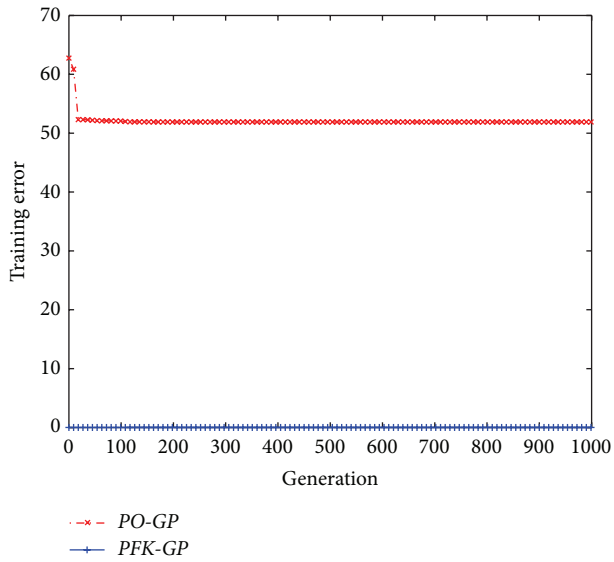

Figure 6: The evolutionary result of P_{13} with *PO-GP* and *PFK-GP*.

Table 7: Training data of algorithm *TrainDBN* (see Algorithm 1).

Number	Formula
P_1	$y = z + a * x_0$
P_2	$y = z + \left(a_1 * \frac{(x_3 + x_1)}{(a_2 * x_4)}\right)$
P_3	$y = z + \left(a_1 * \left(\frac{((x_3 - x_0) + (x_1/x_4))}{(a_2 * x_4)}\right)\right)$
P_4	$y = z + a * \sin(x)$
P_5	$y = z + a * \log(x)$
P_6	$y = z + a * \mathrm{sqrt}(x)$
P_7	$y = z - (a_1 * \exp(a_2 * x_0))$
P_8	$y = z + (a_1 * \mathrm{sqrt}(a_2 * x_0 * x_3 * x_4))$
P_9	$y = \left(\left(\frac{(a_1 * \mathrm{sqrt}(x_0))}{(a_2 * \log(x_1))}\right) * \left(\frac{(a_3 * \exp(x_2))}{(a_4 * \mathrm{square}(x_3))}\right)\right)$
P_{10}	$y = z + \left(a_1 * \left(\frac{((a_2 * x_1) + (a_3 * \mathrm{square}(x_2)))}{(a_4 * \mathrm{cube}(x_3)) + (a_5 * \mathrm{quart}(x_4))}\right)\right)$
P_{11}	$y = z + (a_1 * \cos(a_2 * x_0 * x_0 * x_0))$
P_{12}	$y = z - (a_1 * (\cos(a_2 * x_0) * \sin(a_3 * x_4)))$
P_{13}	$y = z - \left(a * \left(\left(\frac{\tan(x_0)}{\tan(x_1)}\right) * \left(\frac{\tan(x_2)}{\tan(x_3)}\right)\right)\right)$
P_{14}	$y = z - \left(a * \left((\cos(x_0) - \tan(x_1)) * \left(\frac{\tanh(x_2)}{\sin(x_3)}\right)\right)\right)$
P_{15}	$y = z - \left(a * \left(\left(\frac{\tan(x_0)}{\exp(x_1)}\right) * (\log(x_2) - \tan(x_3))\right)\right)$
P_{16}	$y = a * x_3$
P_{17}	$y = a_1 * x_1 + a_2 * x_4$
P_{18}	$y = \frac{\mathrm{sqrt}(x_2)}{\tan(x_5/a)}$
P_{19}	$y = \frac{\cos(x_2)}{\mathrm{cube}(x_5/a)}$
P_{20}	$y = \tanh(x_2 * a * \mathrm{cube}(x_5 + \mathrm{abs}(x_1)))$
P_{21}	$y = \tanh(\mathrm{abs}(x_2 * a + x_5) * \mathrm{cube}(x_5 + \mathrm{abs}(x_1)))$
P_{22}	$y = \tanh\left(\tan\left(\frac{x_5}{a}\right) * \mathrm{cube}(x_5 + \mathrm{abs}(x_1))\right)$
P_{23}	$y = \tanh(\cos(x_2 * a) * \mathrm{cube}(\mathrm{sqrt}(x_2)))$
P_{24}	$y = \tanh(\cos(x_2 * a) * \mathrm{cube}(x_5 + \mathrm{abe}(x_1)))$
P_{25}	$y = z$
P_{26}	$y = z + x_2$
P_{27}	$y = \frac{(z + x_2)}{(x_0 * x_2)}$
P_{28}	$y = \left(\frac{(x_0 - z_1)}{(x_0 + x_2)}\right) * \left(\frac{(x_5 - z_2)}{(x_0 * a_1)}\right)$
P_{29}	$y = a * \mathrm{sqrt}(x)$
P_{30}	$y = a * \log(x)$
P_{31}	$y = a * \mathrm{square}(x)$
P_{32}	$y = a * \tanh(x)$
P_{33}	$y = a * \sin(x)$
P_{34}	$y = a * \cos(x)$
P_{35}	$y = a * \exp(x)$
P_{36}	$y = a * \mathrm{cube}(x)$
P_{37}	$y = a * \mathrm{quart}(x)$
P_{38}	$y = a * \tan(x)$
P_{39}	$y = a * \mathrm{abs}(x)$

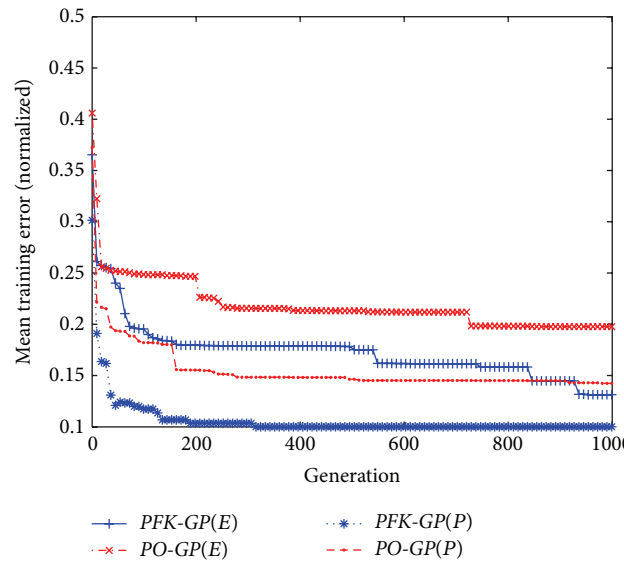

Figure 7: Average results from six groups of means training errors in *PO-GP* and *PFK-GP*.

and *PFK-GP* deal with the SR problem, respectively. The result was illustrated as Figure 6, where

$$\mathrm{Trainerror} = \sqrt{\mathrm{mean}\left((y\mathrm{train} - y\mathrm{predtrain})^2\right)}. \quad (2)$$

We could find that after processing test data of formula model P_{13}, *PFK-GP* found its best model at the first generation and its fitness is higher, while *PO-GP* found its best model until 718th generation and its fitness is much lower than that in *PFK-GP*. The *PFK-GP* can get the right formulas quickly because the model P_{13} recognized by the algorithm *GenerateFs* is inserted into the initialized population of evolutionary computation. For the formula models whose characteristics are consistent with data features in *PFKB*, they can be recognized with high probability and can be combined into population of *PFK-GP*. The *PFK-GP* can firstly search the coefficients in these formula models and get the mathematical expression with good fitness value. Therefore, the algorithm *GenerateFs* can speed up the process of *PFK-GP* dealing with SR and can improve the accuracy of SR results.

In order to test whether *PFK-GP* can overcome overfitting or not, a dataset is created by E_1 which has not existed in the training models of *PFKB*. The two algorithms *PO-GP* and *PFK-GP* are, respectively, applied to process the dataset. The two algorithms, which run, respectively, 100 and 1000 generations, have similar convergence curves in Figure 8. However, *PFK-GP* can find better fitness results compared with *PO-GP*, because *PFK-GP* searches fitted solution in the space includes more functions whose data features are in accord with E_1. Since the initial population, which is generated by the algorithms (*CountSamePartF* and *CountSpecU*) in *PFKB*, contains subformulas in formula models which are recognized by *PFKB* and represents data features of these subformulas, *PFK-GP* can find the right formulas which are more fitted to the raw dataset.

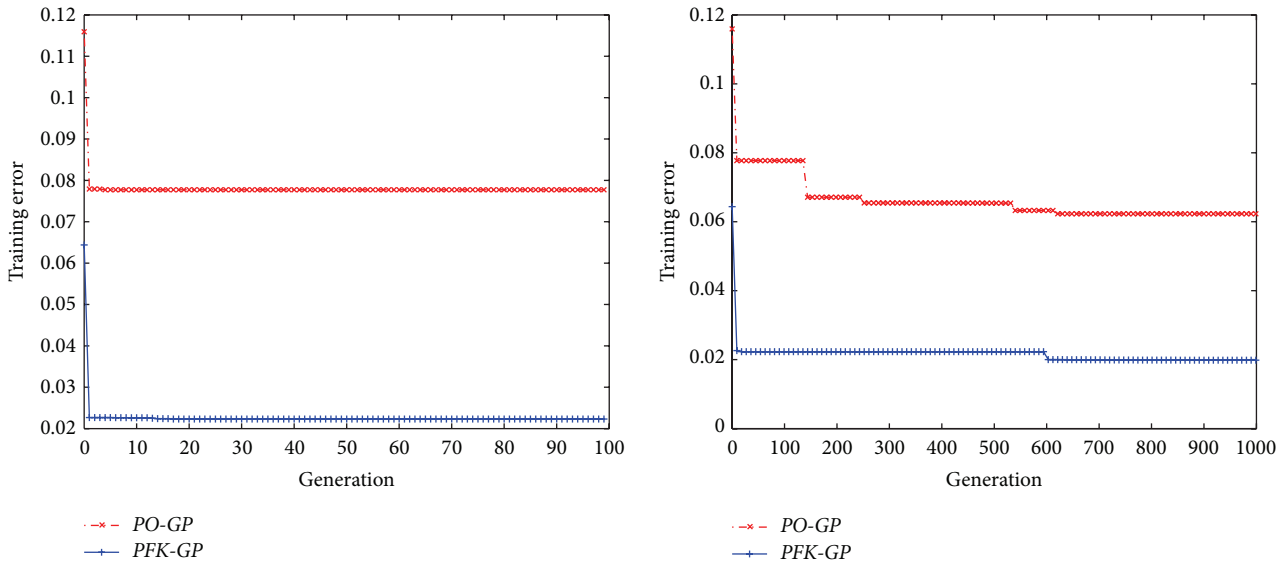

Figure 8: The SR evolutionary process of E_1 with *PO-GP* or *PFK-GP* under different generations.

In order to observe overall performance of the *PFK-GP*, we select six datasets as testing set. Three of them generated by formula models (P_9, P_{13}, P_{19}) from Table 7 are involved in the process of training *DBN*, while the other three generated by formula models (E_1, E_4, E_6) from Table 5 are not involved in that process. The two algorithms *PFK-GP* and *PO-GP* are executed, respectively, ten times in order to gain the right formulas from the six different datasets. The six results of mean training error gained by the two algorithms are shown in Figure 9. And the average results from six groups of mean training errors are listed in Figure 7. The *PFK-GP(E)* and *GP(E)* are the average results of E_1, E_4, and E_6, while *PFK-GP(P)* and *PO-GP(P)* are the average results of P_9, P_{13}, and P_{19}. We can conclude that the comprehensive performance of the *PFK-GP* is better than that of the *PO-GP* based on the results in Figures 7 and 9, because the algorithm *PFK-GP* utilizes the method *GenerateFs* to find the fitted formula model directly and the methods *CountSamePartF* and *CountSpecU* to identify subformula models which have data features consistent with test set. The best mathematical expressions *PFK-GP* and *PO-GP* found are listed in Table 4.

In order to measure relativity between experimental data and predictive data, the formula Training Variation Explained (TVE) is defined as follows:

$$\mathrm{TVE} = 1 - \frac{\mathrm{sum}\left(\left(y\mathrm{train} - y\mathrm{predtrain}\right)^2\right)}{\mathrm{sum}\left(\left(y\mathrm{train} - \mathrm{mean}\left(y\mathrm{train}\right)\right)^2\right)}. \quad (3)$$

The higher the TVE value, the more valid the predictive data. *PO-GP* and *PFK-GP* are run ten times, respectively, in the dataset generated from eight prediction models (see Table 6 E_1–E_8). The eight results of different dataset processed by the above two algorithms are listed in Figure 10. And the maximum, minimum, and average results of TVE are listed in Figure 11. From the results in the two figures, the formulas that *PFK-GP* finds are more relative to the experimental formula models than those *PO-GP* finds.

5. Related Work

The search space of SR is huge even for rather simple basis functions [31]. In order to avoid search space that is too far from the desired output range determined by the training dataset, the interval arithmetic [34] and the affine arithmetic [26], which can compute the bounds of *GP* tree expression, are imported into SR. Although the method based on affine arithmetic can generate the tighter bounds of the expression in comparison with the interval arithmetic method, its accuracy often leads to high computational complexity [35]. Moreover, the size of search space is still huge because there are plentiful candidate expressions which fit to the data bound computed by the above two arithmetic methods.

In addition to the above arithmetic method, machine learning methods are used to compact or reduce the search space of SR. FFX technology uses pathwise regularized learning algorithm to rapidly prune a huge set of candidate basis functions down to compact model based on the generalization linearly model (GLM); hence the technology outperforms GP-SR in speed and scalability due to its simplicity and deterministic nature [8]. However, it may abandon correct expressions and make them not in the space of GLM. A hybrid deterministic GP-SR algorithm [36] is proposed to overcome the problem of missing correct expression. The hybrid algorithm extracts candidate basis expressions by using FFX and inputs the expressions into the GP-SR. The hybrid algorithm utilizes the candidate expression generated by the linear regression method (pathwise regulation), while our algorithm utilizes the candidate expression by applying the algorithms *CountSamePartF, GenerateFs,* and *CountSpecU*.

By applying expectation-maximization (EM) framework to SR, the clustered SR (CSR) can identify and infer symbolic repression of piecewise function from unlabelled, time-series data [9]. The CSR can reduce the space of searching piecewise function owing to the fact that the EM can search

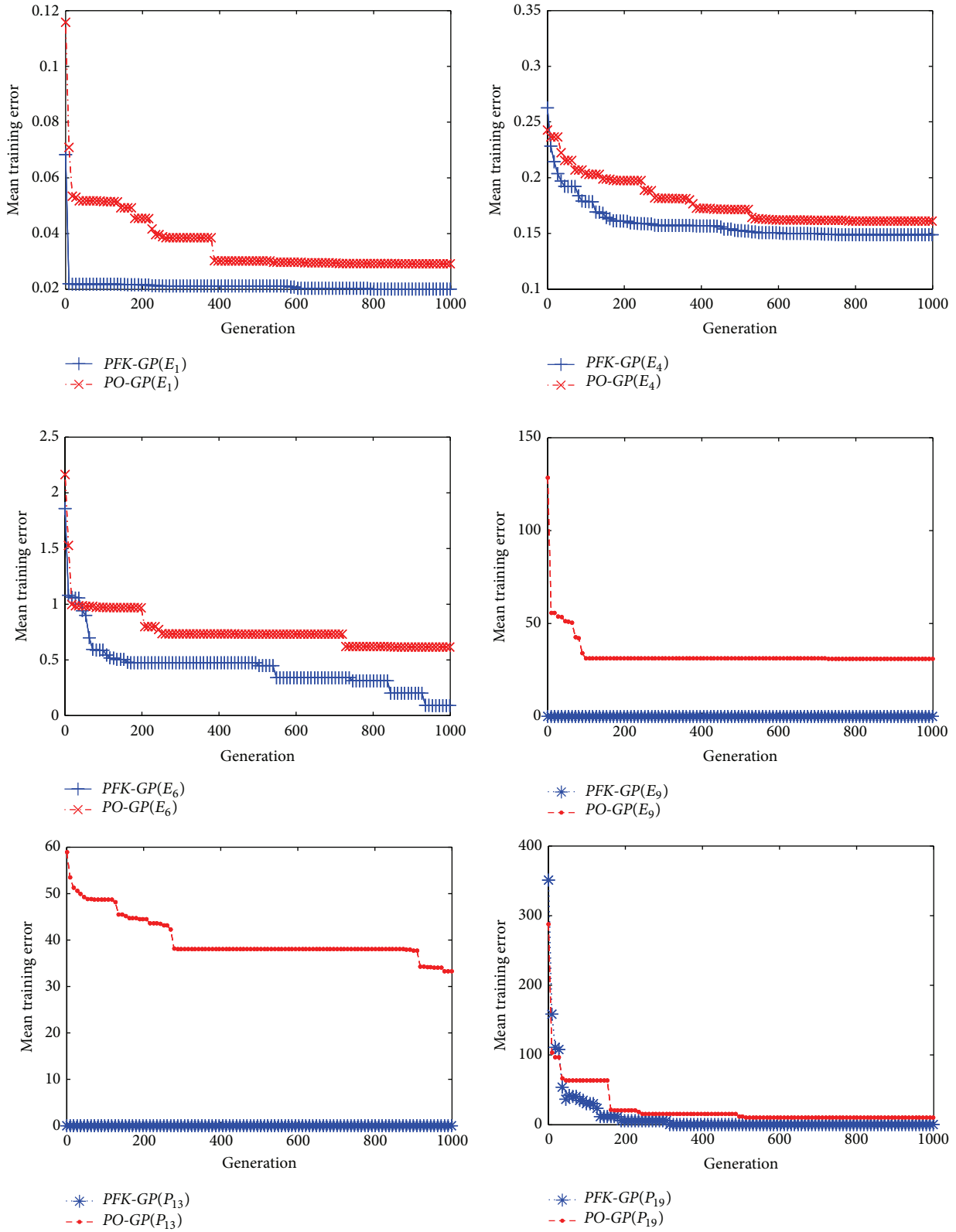

Figure 9: Training error results in which six datasets generated by E_1, E_4, E_6, P_1, P_{13}, and P_{19} dealt with *PO-GP* and *PFK-GP*, respectively.

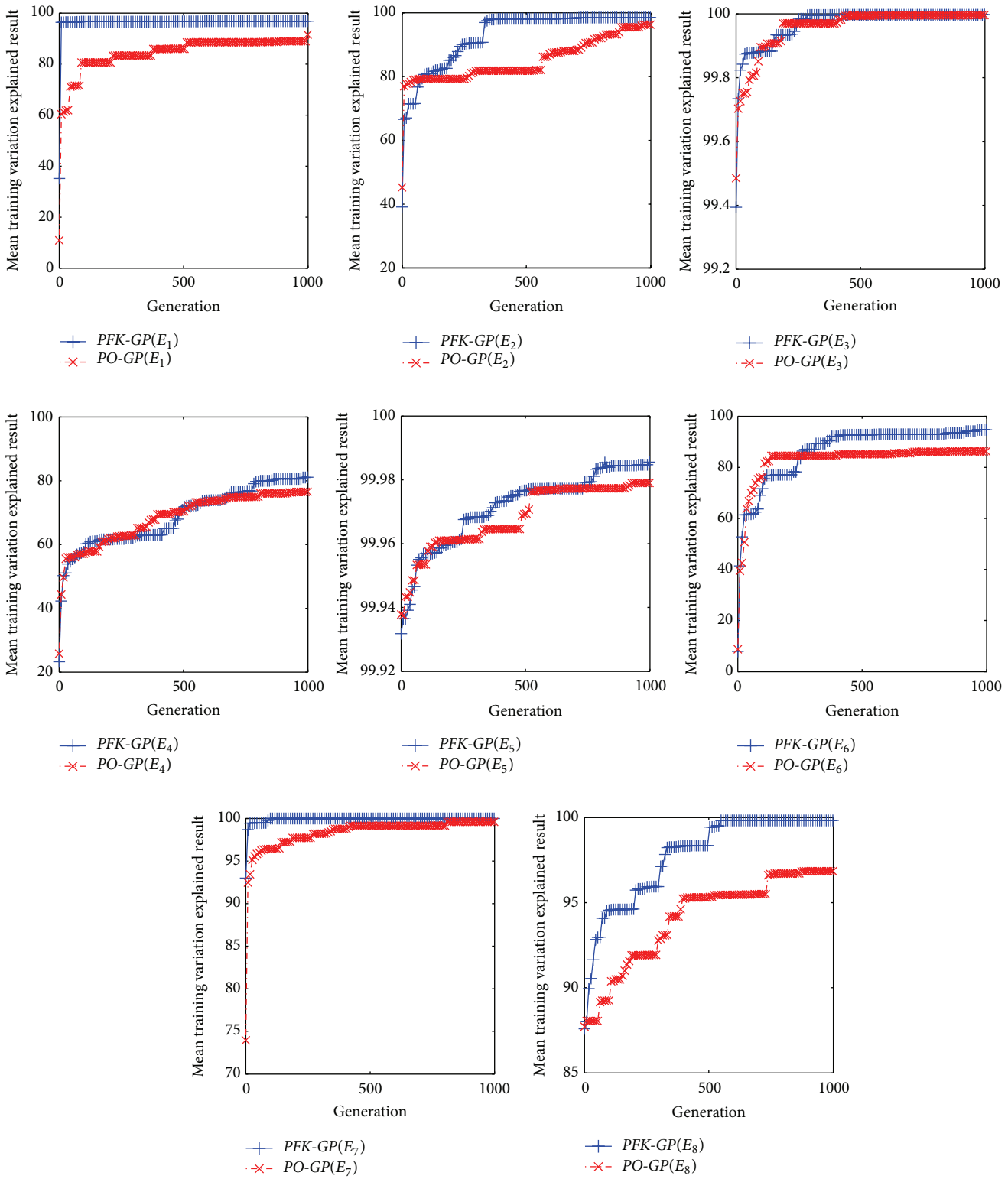

Figure 10: TVE for models E_1–E_8 contrast the random population in traditional *PO-GP* with the *PFK-GP*.

simultaneously the subfunction parameters and latent variables that represent the information of function segment. The abstract expression grammar (AEG) SR is proposed to perform the process of genetic algorithm (GA), allowing user control of the search space and the final output formulas [37]. On understanding the given application, users can specify the goal expression of SR and limit the size of search space by using abstract expression grammars. Compared with manually assigning expression and limiting the search space with AEGSR, in the paper, the methods about PFK can

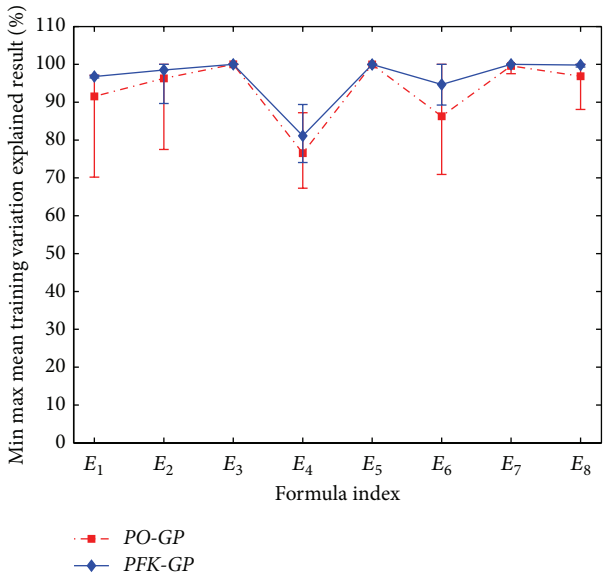

Figure 11: TVE results of *PO-GP* compared with *PFK-GP* in eight formula models.

automatically extract the candidate expression from dataset by using statistical method and dynamically adjust the search space by using *GP*.

The methods that inject prior or expert knowledge in evolutionary search [12, 13] are introduced to find effective solutions that can show mathematical expression more compactable and interpretable. In these papers, the prior and expert knowledge are the solutions which are mathematical expressions in some applications. The knowledge is merged into *GP* by inserting randomized pieces of the approximate solution into population. One of the major differences between these methods and our method is how prior or expert knowledge is created. The knowledge in [12, 13] is the existing formula model that comes from the previous solutions and can be called static knowledge. However, the knowledge in our method is the formula model which is consistent with data features that are originated from the algorithms *GenerateFs*, *CountSamePartF*, and *CountSpecU* and can be called dynamical knowledge that is changed with the features of test dataset. Therefore, our methods can insert more suitable knowledge into the *GP*.

6. Conclusion

In this paper, a *PFK-GP* method is proposed to deal with the problem of symbolic regression based on analyzing the process of how a researcher constructs a mathematical model. The method can understand experimental data features and can extract some formulas consistent with experimental data features. In order to implement the function of understanding data features, *PFK-GP*, through the *DBN* method, firstly creates *PFKB* that can extract features from test dataset generated by training formula models. The experiment results confirm, compared with SVM, that *DBN* can produce better results that extract features from formula models and classify test data into its corresponding formula model. Then, the methods of classification and recognition are implemented to find some formula models that are similar or related to experimental data features as much as possible. For the classification, we exploit the algorithm *GenerateFs* based on *DBN* to match the experimental data with formula models in *PFKB*. With regard to recognition, we propose the algorithms of *CountSamePartF* and *CountSpecU* to obtain some subformula models which have local features consistent with experimental data. The classification can help *PFKB* to find formula models that are consistent with whole data features while the recognition can help *PFKB* to find subformula models consistent with local data features. At last, the algorithm *randomGenP* is used to generate individuals of evolutionary population according to the result of the above three algorithms. Through combining and transforming these individuals, *GP* can automatically obtain approximate formulas that are best fitting to the experimental data.

Compared with Pareto *GP*, *PFK-GP*, which is built on the *PFKB* with the functions of classification and recognition, can explore formulas in the search space of data features. So, it can accelerate the speed of convergence and improve the accuracy of formula obtained.

Obviously, the high efficiency of *PFK-GP* depends on the powerful methods of classification and recognition based on *PFKB*. Therefore, it is an important part of the future work to improve the accuracy of the above two methods. The two methods depend on the representation of data features of formula model. In the paper, the two assumptions based on statistics and counts are used to obtain the formulas which can show the data features. The features of formula model are not defined explicitly. And the two assumption are not proved by formal proofs. There are some uncertainties in those assumptions. Therefore, the new representation which can show whole or local features of formula models will be researched to find formulas which can better fit to experiment data. In addition, the rules of formulas transforming and inferring that are similar to researchers' methods will be explored in the evolution of *GP*.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work was supported by National Natural Science Foundation of China (Grant no. 61402532), Science Foundation of China University of Petroleum-Beijing (no. 01JB0415), and China Scholarship Council.

References

[1] M. Schmidt and H. Lipson, "Distilling free-form natural laws from experimental data," *Science*, vol. 324, no. 5923, pp. 81–85, 2009.

[2] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, Mass, USA, 1992.

[3] C. Ferreira, "Gene expression programming: a new adaptive algorithm for solving problems," *Complex Systems*, vol. 13, no. 2, pp. 87–129, 2001.

[4] M. O'Neill and C. Ryan, "Grammatical evolution," *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 4, pp. 349–358, 2001.

[5] J. O'Sullivan and C. Ryan, "An investigation into the use of different search strategies with grammatical evolution," in *Genetic Programming*, J. Foster, E. Lutton, J. Miller, C. Ryan, and A. Tettamanzi, Eds., vol. 2278 of *Lecture Notes in Computer Science*, pp. 268–277, Springer, Berlin, Germany, 2002.

[6] Z. Oplatkova and I. Zelinka, "Symbolic regression and evolutionary computation in setting an optimal trajectory for a robot," in *Proceedings of the 18th International Workshop on Database and Expert Systems Applications (DEXA '07)*, pp. 168–172, IEEE, Regensburg, Germany, September 2007.

[7] X. Yao, Y. Liu, and G. Lin, "Evolutionary programming made faster," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 2, pp. 82–102, 1999.

[8] T. McConaghy, "Ffx: fast, scalable, deterministic symbolic regression technology," in *Genetic Programming Theory and Practice IX*, R. Riolo, E. Vladislavleva, and J. H. Moore, Eds., Genetic and Evolutionary Computation, pp. 235–260, Springer, New York, NY, USA, 2011.

[9] D. L. Ly and H. Lipson, "Learning symbolic representations of hybrid dynamical systems," *Journal of Machine Learning Research*, vol. 13, pp. 3585–3618, 2012.

[10] P. Tomson and G. W. Greenwood, "Using ant colony optimization to find low energy atomic cluster structures," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '05)*, vol. 3, pp. 2677–2682, September 2005.

[11] M. O'Neill, L. Vanneschi, S. Gustafson, and W. Banzhaf, "Open issues in genetic programming," *Genetic Programming and Evolvable Machines*, vol. 11, no. 3-4, pp. 339–363, 2010.

[12] M.-R. Akbarzadeh-T and M. Jamshidi, "Incorporating a-priori expert knowledge in genetic algorithms," in *Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA '97)*, pp. 300–305, IEEE, Monterey, Calif, USA, July 1997.

[13] M. D. Schmidt and H. Lipson, "Incorporating expert knowledge in evolutionary search: a study of seeding methods," in *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation (GECCO '09)*, pp. 1091–1098, ACM, Montreal, Canada, July 2009.

[14] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[15] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

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

[17] G. Smits and M. Kotanchek, "Pareto-front exploitation in symbolic regression," in *Genetic Programming Theory and Practice II*, vol. 8 of *Genetic Programming*, pp. 283–299, Springer, 2005.

[18] J. Neter, W. Wasserman, and M. Kutner, *Applied Linear Regression Models*, Irwin, Martinsville, Ohio, USA, 1989.

[19] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[20] Y. Freund and R. Schapire, "A desicion-theoretic generalization of online learning and an application to boosting," in *Computational Learning Theory*, vol. 904 of *Lecture Notes in Computer Science*, pp. 23–37, Springer, Berlin, Germany, 1995.

[21] H. Chen, P. Tiňo, and X. Yao, "Probabilistic classification vector machines," *IEEE Transactions on Neural Networks*, vol. 20, no. 6, pp. 901–914, 2009.

[22] G. Hinton, L. Deng, D. Yu et al., "Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[23] R. Collobert and J. Weston, "A unified architecture for natural language processing: deep neural networks with multitask learning," in *Proceedings of the 25th International Conference on Machine Learning (ICML '08)*, pp. 160–167, ACM, July 2008.

[24] R. Poli and N. F. McPhee, "Parsimony pressure made easy," in *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation (GECCO '08)*, pp. 1267–1274, ACM, Atlanta, Ga, USA, July 2008.

[25] M. Schmidt and H. Lipson, "Age-fitness pareto optimization," in *Genetic Programming Theory and Practice VIII*, R. Riolo, T. McConaghy, and E. Vladislavleva, Eds., vol. 8 of *Genetic and Evolutionary Computation*, pp. 129–146, Springer, New York, NY, USA, 2011.

[26] C. L. Pennachin, M. Looks, and J. A. de Vasconcelos, "Robust symbolic regression with affine arithmetic," in *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation (GECCO '10)*, pp. 917–924, ACM, Portland, Ore, USA, July 2010.

[27] Y. Freund and D. Haussler, "Unsupervised learning of distributions on binary vectors using two layer networks," Tech. Rep. UCSC-CRL-94-25, 1994.

[28] Y. Bengio, Y. Lecun, and Y. Lecun, "Convolutional networks for images, speech, and time-series," in *The Handbook of Brain Theory and Neural Networks*, pp. 255–258, AT&T Bell Laboratories, 1995.

[29] D. Searson, "Gptips package for matlab," 2012, http://sites.google.com/site/gptips4matlab.

[30] R. Palm, "Deepldeeptoolbox," 2012, https://github.com/rasmusbergpalm/DeepLearnToolbox.

[31] M. F. Korns, "A baseline symbolic regression algorithm," in *Genetic Programming Theory and Practice X*, R. Riolo, E. Vladislavleva, M. D. Ritchie, and J. H. Moore, Eds., Genetic and Evolutionary Computation, pp. 117–137, Springer, New York, NY, USA, 2013.

[32] Toy benchmarks test suite 1, http://www.symbolicregression.com/?q=node/5.

[33] C.-C. Chang and C.-J. Lin, "LIBSVM: a Library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, no. 3, article 27, 2011.

[34] M. Keijzer, "Improving symbolic regression with interval arithmetic and linear scaling," in *Genetic Programming*, C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, and E. Costa, Eds., vol. 2610 of *Lecture Notes in Computer Science*, pp. 70–82, Springer, Berlin, Germany, 2003.

[35] P. Baranyi, Y. Yam, D. Tikk, and R. J. Patton, "Trade-off between approximation accuracy and complexity: TS controller design via HOSVD based complexity minimization," in *Interpretability Issues in Fuzzy Modeling*, J. Casillas, O. Cordn, F. Herrera, and L. Magdalena, Eds., vol. 128 of *Studies in Fuzziness and Soft Computing*, pp. 249–277, Springer, Berlin, Germany, 2003.

[36] I. Icke and J. C. Bongard, "Improving genetic programming based symbolic regression using deterministic machine learning," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '13)*, pp. 1763–1770, IEEE, Cancun, Mexico, June 2013.

[37] M. Korns, "Abstract expression grammar symbolic regression," in *Genetic Programming Theory and Practice VIII*, R. Riolo, T. McConaghy, and E. Vladislavleva, Eds., vol. 8 of *Genetic and Evolutionary Computation*, pp. 109–128, Springer, New York, NY, USA, 2011.