GEP-NFM: Nested Function Mining Based on Gene Expression Programming

Taiyong Li1,2, Changjie Tang1, Jiang Wu1,2, Xuzhong Wei1, Chuan Li1, Shucheng Dai1, Jun Zhu3

1 School of Computer Science, Sichuan University, Chengdu, Sichuan, China, 610065
2 School of Economic Information Engineering, Southwestern University of Finance and Economics, Chengdu, Sichuan, China, 610074
3 Birth Defects Supervising Centre of Western China Medical School, Sichuan University, Chengdu 610065, China

{litaiyong, tangchangjie, wujiang, weixuzhong, lichuan, daishucheng}@cs.scu.edu.cn
lity_f@swufe.edu.cn
zhu_jun1@163.com

Abstract

Mining the interesting functions from the large scale data sets is an important task in KDD. Traditional gene expression programming (GEP) is a useful tool to discover functions. However, it cannot mine very complex functions. To resolve this problem, a novel method of function mining is proposed in this paper. The main contributions of this paper include: (1) analyzing the limitations of function mining based on traditional GEP, (2) proposing a nested function mining method based on GEP (GEP-NFM), and (3) experimental results suggest that the performance of GEP-NFM is better than that of the existing GEP-ADF. Averagely, compared with traditional GEP-ADF, the successful rate of GEP-NFM increases 20% and the number of evolving generations decrease 25%.

1. Introduction

Discovering the interesting schema in large scale data sets is an important and challenge task in machine learning and knowledge discovery. Mining the function relationship is such a task. Traditional gene expression programming (GEP) can mine only relatively simple functions. In this study, we propose a novel method GEP-NFM to mine complex functions with high successful rate.

The rest of this paper is organized as follows. Section 2 describes the related work. Section 3 introduces the preliminary terminology and concepts in GEP. Section 4 proposes the novel method based on GEP to mine functions. Section 5 provides experimental evaluations. Finally, Section 6 concludes this paper.

2. Related work

Function mining is a hot research topic in gene expression programming. Literature [1] proposed a method to mine functions in relatively simple form constructed by primary mathematical operators, such as +, -, *, /, and \. In reference [2], automatically defined functions (ADFs) were introduced and ADFs were treated as terminals in the chromosome. The ADFs were usually generated randomly and may be irrelevant completely to the real functions. In ADFs, one function cannot call another function as a parameter. These limitations make obstacle for mining complex functions.

In this paper, we propose a novel method of function mining called GEP-NFM. The main idea is introducing nest-able basic functions (BFs), that is, one BF can call another BF and that the BF is treated not as a terminal symbol but a function symbol. Therefore, several BFs can be used to build more complex functions.

3. Preliminary terminology and concepts in GEP

GEP is of genetic computing proposed by Candida in 2001[2]. GEP combines the advantages of both GA and genetic programming (GP), while overcoming some of their individual limitations. The chromosome of GP is tree-formed structure directly, while that of GEP is linear string. So the genetic operations of GP are designed to manipulate the tree forms of chromosomes. However, the genetic operations of GEP are similar to but simpler than those in GA. Compared with its ancestors, GEP innovated in structure and method. It surpasses the GA and GP algorithms by a factor of 100-60,000 in evolution performance [2-6].

The chromosome in GEP is encoded in a linear string of fixed length, and it is composed of one or more genes. Each gene is divided into a head and a tail. For each problem, the length of gene head h is chosen, whereas the length of the tail t is a function of h and the number of arguments of the function with more arguments n and is evaluated by the equation [2]:

\[ t = h(n-1) + 1 \] (1)

In GEP, the gene head contains symbols that can be chosen from both functions and terminals, whereas the
tail contains only terminals. This special structure feature makes the available length of gene (or chromosome) retractile. Owing to the retractile structure feature, GEP can solve more complicated problem in huge search space.

In decoding stage, the chromosome is mapped into an expression tree(ET). After mapped into ET, a chromosome can be transformed into a mathematical expression.

**Example 1.** Let the functions set FS= \{+,\ast , /\}, the terminals set TS= \{x, y\}, the length of head is 5 and the number of gene is 1. Figure 1 shows the encoding and decoding process in GEP.

```
a. 0 1 2 3 4 5 6 7 8 9 0
   / + - * * x y x x y y
b. (x^2 + y^2)/(x-y)
```

**Figure 1. ET and Expression for chromosome. a) The chromosome of the individual. b) The ET encoded by the chromosome. c) The expression decoded by the ET.**

GEP has been applied in wide application areas, such as function mining, time series predication, association rule mining, classification, neural network and so on\[^2,6-11\].

### 4. Nested Function Mining based on GEP: GEP-NFM

#### 4.1 Constructing basic functions (BFs)

Most complex functions can be decomposed into several simple factors. For example, the polynomial \( f(x) = x^6 - 2x^4 + x^2 \) can be factorized as follows:

\[
f(x) = x^6 - 2x^4 + x^2 = x^2(x-1)^2(x+1)^2
\]  

One of the possible forms to express \( f(x) \) is shown below:

\[
g(x) = x^2 \quad (3) \]

\[
h(x) = x-1 \quad (4) \]

\[
p(x) = x+1 \quad (5)
\]

It is easy to verify that

\[
f(x) = g(x)\ast h(x)\ast p(x) \quad (6)
\]

Equations (3)-(5) are called basic functions (BFs). These functions can build more complex functions. Usually, these basic functions can be defined by priori knowledge. If these functions are generated randomly, they become automatically defined functions as described in \[^2\].

BF is usually composed of basic mathematical operators and some other BF's. To build complex functions, several BF's are needed.

#### 4.2 Encoding

Our model of chromosome has 2 parts, i.e., the basic functions and the cells.

Different from other encodings, our encoding method does not treat BF as a terminal but as a function.

BFs are encoded with basic mathematical operators and the other BF's and terminals.

When encoding BF, we use \( h_B \) and \( t_B \) to denote the head and tail length of BF respectively. We also use \( h_C \) and \( t_C \) to denote the head and tail length of cell respectively. To explain the main idea of encoding simply, we assume that (a) each chromosome has three BF's and two cells, (b) each BF has only one parameter and BF can call the other BF's but can not call itself. If one BF calls itself, it will be a recursive function. We also use 0,1,2 to denote the first, the second, and the third BF ,i.e., BF_0-BF_2 respectively.

**Example 2.** Let \{+,\ast ,/\},0,1,2\} be the function set, \{x\} be the terminal set. \( h_B=4,t_B=5,h_C=6,t_C=7 \). Therefore, the total length of the chromosome is \((4+5)+3+(6+7)\ast 2=53\). Cell_0 and Cell_1 are linked by \( \ast \). One chromosome is shown in Figure 2 a. In this chromosome, Cell_0 and Cell_1 include 0,1 or 2, which means BF_0,BF_1, BF_2 are used in cells. The decoding procedure is shown in Figure 2 b-d.

```
a. 0 1 2 3 0 1 2 3 4  0 1 2 3 0 1 2 3 4   0 1 2 3 0 1 2 3 4
   * x x + x x x x x  – x  / x x x x x x   0 1 x + /  x x x x
   BF_0 BF_1 BF_2
b. 0 1 2 3 0 1 2 3 4  0 1 2 3 4 5 0 1 2 3 4 5 6  0 1 x / x x x x x
   * x x x x x x x + / x x x x x x x x x x
   BF_0 BF_1 BF_2
```

**Figure 2.** a. In this chromosome, Cell_0 and Cell_1 include 0,1 or 2, which means BF_0,BF_1, BF_2 are used in cells. The decoding procedure is shown in Figure 2 b-d.
The cells encoded in the
−  \(j\) (7). Let \(M\), \(N\) and \(K\) be the maximum \(V\).
\(\subseteq\) \(V\)                                    (8)
fitness cases, if \(P_{ij}\), and \(n\) is the number of fitness cases. Obviously, for all
and it means the individual program \(i\) is ideal.

\[f_i = \frac{1}{\sum_{j=1}^{n} \left( \frac{P_{ij} - T_j}{T_j} \right)^2 + 1} \tag{7}\]
where \(P_{ij}\) is the value predicted by the individual program \(i\) for fitness case \(j\) and \(T_j\) is the target value for fitness case \(j\), and \(n\) is the number of fitness cases. Obviously, for all fitness cases, if \(P_{ij} = T_j\), then \(f_i\) equals the maximum, i.e. 1, and it means the individual program \(i\) is ideal.

4.4 Genetic operators

The elitism and roulette-wheel sampling strategies are used to select individual for next generation as following. (a) The best individual (with highest fitness), i.e. elitism., is passed on the next generation directly.(b)If there are several elitisms, the last one will be selected to pass on to the next generation.(c) For the individuals which are not the elitisms, they will be selected according to fitness by roulette-wheel sampling.

Mutation and recombination are also used to keep the diversity of the population in our model.

The steps of mutation are as below: (a) For each individual, deciding if it should mutate or not according to mutate rate \(P_m\); (b) For the individual to mutate, generating a random position of mutation; (c) If the position is in the heads of genes, any symbol can change into another symbol (basic mathematical operator, BF or terminal); and if it is in the tails, terminals can only change into terminals.

The steps of recombination are as below: (a) For two individuals, deciding if they should recombination or not according to recombination rate \(P_r\); (b) For the two individuals to recombination, generating random positions of recombination for each BF or cell; (c) Exchanging the symbols after the positions of each BF or cell.

4.5 GEP-NFM algorithm

The GEP-NFM algorithm to improve the performance of function mining based on GEP is illustrated as below.

**GEP-NFM Algorithm**: Nested Function Mining Based on GEP

**Input**: Cases, \(N\) (population size), \(P_m\) (mutate rate), \(P_r\) (recombine rate), \(M\) (maximum of generations), \(N_{\text{max}}\) (the maximum generations of the best individual does not improve continuously)

**Output**: \(B_I\) (the best individual to fit the cases)

1. \(S\leftarrow\text{Initial Population;}
2. \(B_I\leftarrow\text{Null; } n\leftarrow 0;
3. \text{Repeat}
4. \text{Calculate the fitness value of each individual according to Equation(7);}
5. \(f\leftarrow\text{the maximum fitness value of generation } n;\)
6. \(B_I\leftarrow\text{the best individual of generation } n;\)
7. \(n++;\)
8. \(S\leftarrow\text{Mutate } S \text{ by } P_m;\)
9. \(S\leftarrow\text{Recombine } S \text{ by } P_r;\)
10. Until ((\(n>=M\)) \&\& (\(B_I\) not improvement in \(N_{\text{max}}\) generations)) \&\&(f= =1))
11. return \(B_I;\)
End.

There are three terminal conditions of iterations. (a)The evolution generation exceeds \(M\) (maximum of generations). (b)The best individual does not improve in continuous \(N_{\text{max}}\) generations. (c) The fitness of the best individual is very close to 1.

**Proposition 1**. Let \(M\), \(N\) and \(K\) be the maximum evolving generations, population size and case size respectively. And let \(N_{\text{max}}\) be the maximum generations of the best function expression does not improve continuously. In the worst case, the complexity of GEP-NFM is \(O(N\times N_{\text{max}}\times M)\). In the best case, the complexity of GEP-NFM is \(O(N\times K\times N_{\text{max}})\).

**Proof**. In GEP-NFM, the complexity of calculating an individual’s fitness for \(k\) cases is \(O(K)\). For a population, the complexity of calculating all individuals’ fitness is \(O(N\times K)\). In the worst case, the times of the iterations is \(M\), thus, the complexity is \(O(N\times K\times M)\). In the best case, the times of the iterations is \(N_{\text{max}}\), thus, the complexity is \(O(N\times K\times N_{\text{max}})\).

**Proposition 2**. Let \(U\), \(V\) be the function set discovered by GEP-ADF and GEP-NFM respectively. Then \(U \subseteq V\).

**Proof**. In GEP-NFM, one BF can call the other BFs. When each BF does not call any other BFs, GEP-NFM degenerates into GEP-ADF. Therefore, GEP-ADF is a special case of GEP-NFM, i.e.,

\[U \subseteq V\] (8)
5. Experiments and performance analysis

This section reports implementation of the algorithm and analyzes its performance compared with the GEP-ADF.

5.1 Experimental setup

The parameters for the experiments are shown in Table 1. We compare GEP-NFM with GEP-ADF.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>GEP-NFM</th>
<th>GEP-ADF</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of runs</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>No. of generations</td>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td>Population size</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>Function set</td>
<td>+,-,*/BF0.2</td>
<td>+,-,*/</td>
</tr>
<tr>
<td>Terminal set</td>
<td>x,y</td>
<td>x,y, ADF0.2</td>
</tr>
<tr>
<td>BF/ADF length</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>No. of BFs/ADFs</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Chromosome length</td>
<td>60</td>
<td>60</td>
</tr>
<tr>
<td>Mutation rate(Pr)</td>
<td>0.017</td>
<td>0.017</td>
</tr>
<tr>
<td>Recombination rate(Pr)</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>Fitness function</td>
<td>Equation (7)</td>
<td>Equation (7)</td>
</tr>
<tr>
<td>Nmax</td>
<td>20</td>
<td>20</td>
</tr>
</tbody>
</table>

The experiments were conducted on a PC with Intel Pentium 4 2.0GHz CPU and 1G main memory, and the operation system is Windows XP Professional. The development environment is Microsoft Visual C++ 6.0.

5.2 Experimental results

**Experiment 1.** Let the objective function be as follows:

\[ f(x, y) = x^3 + xy + 5x + 4y + 5 \]  \( (9) \)

By (9), we choose randomly 20 points in the real interval \([-20,20]\) to build the training set \(\{(x_i, y_i, f_i) | i=1,2, \ldots, 20\}\). The results of Experiment 1 are shown in Table 2.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Successful rate</th>
<th>Average generations</th>
</tr>
</thead>
<tbody>
<tr>
<td>GEP-ADF</td>
<td>100%</td>
<td>42</td>
</tr>
<tr>
<td>GEP-NFM</td>
<td>100%</td>
<td>31</td>
</tr>
</tbody>
</table>

In this experiment, the successful rate of GEP-NFM equals that of GEP-ADF, while the average evolving generations are less than those of GEP-ADF. The fitness of the best individual and average fitness of GEP-NFM and GEP-ADF is shown in Figure 3.

**Experiment 2.** Let the objective function be as follows:

\[ f(x, y) = x^6 + 6x^4y + 12x^2y^2 + 8y^3 + x^2 - 2xy \]  \( (10) \)

We also choose randomly 20 points in the real interval \([-20,20]\) to build the training set. The results of the Experiment 2 are shown in Table 3.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Successful rate</th>
<th>Average generations</th>
</tr>
</thead>
<tbody>
<tr>
<td>GEP-ADF</td>
<td>81%</td>
<td>76</td>
</tr>
<tr>
<td>GEP-NFM</td>
<td>100%</td>
<td>49</td>
</tr>
</tbody>
</table>

Table 2 and Table 3 show that GEP-NFM can mine all functions that mined by GEP-ADF. The experiments verified the conclusion in Proposition 2. Furthermore, experiments show that the average generations of GEP-NFM are less than those of GEP-ADF.

6. Conclusions

It is difficult to mine complex functions for traditional GEP algorithm. This paper proposes a novel function mining method. The key technique is that: a complex function is composed of several simple functions, and the simple functions call each other. The authors propose a novel encoding method that views basic function as a function and it is only in the head of the gene as well as a new fitness function. The experiments show that the method has better performance than the existing method and it can mine more complex function while the average generations are less than those of the existing methods.

A lot of work is to be done in the future, such as extending the methods to other application areas, improving the performance of the algorithm and finding better a priori knowledge to construct the basic functions.

References


