基因的无树评估

李 川[1],唐常杰[1],陈 瑜[1],代术成[1],邱江涛[1],罗 谦[1],朱 军[2]
LI Chuan[1],TANG Chang-jie[1],CHEN Yu[1],DAI Shu-cheng[1],QIU Jiang-tao[1],LUO Qian[1],ZHU Jun[2]

1. 四川大学 计算机学院,成都 610065
2. 中国出生缺陷监测中心,成都 610065
1. School of Computer,Sichuan University,Chengdu 610065,China
2. Birth Defects Supervising Centre,Chengdu 610065,China
E-mail:tangchangjie@cs.scu.edu.cn

LI Chuan,TANG Chang-jie,CHEN Yu,et al. Evaluating genes without expression trees construction. Computer Engineering and Applications,2008,44(14):80-84.

Abstract: Traditional ET(Expression Tree)-based GEP has a major performance defect:Repetitive ET traverses and calculations. This paper proposes a novel model called Scale-Based GEP to solve the problem. Variable matrix is used to avoid repetitive calculations and traversing in ET-based gene evaluation. Experiments show that Scale-based evaluation outperforms ET-based method 3~5 times constantly.
Key words: data mining;gene expression programming;Scale-based gene expression

摘 要:传统基于表达式树 ET 的基因评估从性能角度讲主要缺点是:重复遍历表达式树和进行大量重复计算。提出基于 Scale 的基因评估。变量矩阵用来避免基因评估中的重复计算。实验表明,在绝大多数数据分布下和参数选择情况下:基于 Scale 的基因评估较基于 ET 的基因评估快 3~5 倍。
关键词:数据挖掘;基因表达式编程;基于 Scale 的基因表达
DOI:10.3778/j.issn.1002-8331.2008.14.022 **文章编号**:1002-8331(2008)14-0080-05 **文献标识码**:A **中图分类号**:TP311.131

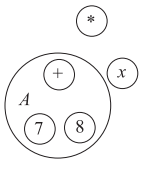
1 Introduction

Gene Expression Programming(GEP) proposed by Candida Ferria is one of the latest advances in the family of evolutionary computation and has wide applications in data mining[1-5]. However,current GEP has a major defect greatly affecting the evolutionary performance.

Major Defect For each fitness instance,ET needs to be evaluated once,which brings severely repetitive traverses and calculations. To explain this problem,see another example.

Example 1 As is shown in Fig. 1,for each fitness instance,the sub-tree in region *A* needs to be traversed and calculated one time. Suppose there are totally 1 000 fitness instances. So totally region *A* will be traversed and calculated 1 000 times. But in fact,only once is enough for all the instances because region *A* always results in (7+8)=15. Let one node operation(visit or calculation) be one fundamental operation. The repetitive traverses and calculations take (3+1)*(1 000-1) fundamental operations,i.e. around 57% of the total fundamental operations((5+2)*1 000)).

To solve the two performance defects this paper did the following contributions. (1) A novel structure,Scale,is proposed for gene expression. Variable matrix is proposed to assistant in the gene evaluation. (2) Algorithm SEVM is proposed to evaluate genes without repetitive operations. (3) Experiments show that Scale-based gene evaluation is 3~5 times faster than the current ET-based method.



(a)An ET

#	x	f(x)
1	2	31.5
2	4	60.5
3	3	44.5
4	7	103.5
…		

(b)Fitness instances

Fig. 1 Repetitive traverses and calculations
图1 重复遍历和重复计算

2 Preliminaries

GA(Genetic Algorithm) and GP(Genetic Programming) are two major evolutionary computation models[6-8]. GEP makes the

the National Natural Science Foundation of China under Grant No. 90409007(国家自然科学基金);the 11th Five Years Key Programs for Sci. & Tech. Development of China under grant No. 2006038002003(国家"十一五"科技支撑项目);the Tackle Key Problem Fund of Sichuan Province under Grant No. 2006Z01-027(四川省科技攻关项目);Sichuan University Youth Foundation under Grant No. 06036(四川大学青年基金)。
Received:2008-01-18 Accepted:2008-03-25

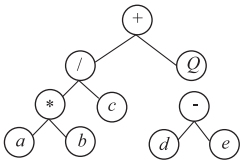
difference by incorporating the advantages of both GA and GP. (1) GEP adopts GA's fixed length gene coding method to maintain simplicity and the easiness of mutation. (2) GEP expresses genes into trees to provide enough capability to describe complex problems. GEP has wide applications ranging from associations mining to clone selection[9-12].

GEP expresses individuals into Expression Trees (ETs). To avoid dead genes generation, all genes abide by the form of "*K*-expression". For the binary operators, if gene head length is h, its tail length should be $h+1$ as shown in Formula (1).

0 1 2 3 4 5 6 7 8 9

$+/Q*c-a\ b\ d\ e$ (1)

By putting the gene elements from top to down, from left to right, the gene can be easily expressed into a tree. Gene(1) is expressed into the ET(Fig. 2(a)).



(a)ET

#	a	b	c	d	e	f(a,b,c,d,e)
1	7	2	3	5	4	5.5
2	5	7	5	5	1	8.7
3	9	4	2	8	7	18.5
4	6	3	3	2	1	7
.....						

(b)Fitness case

$$fitness = \frac{1}{\frac{\sum_{j=1}^{n}\sqrt{|g_j - f_j|}}{n} + 1}$$

(c) Fitness function

Fig. 2　The Expression Tree(ET), the fitness instances and the fitness function

图2　表达式树(ET),适应度实例,适应度函数

3　Scale-Based Genes Evaluation

To overcome the problem of dynamically generating and releasing ET(Defect 1), a novel structure, scale, is introduced.

3.1　The Introduction of Scale

Definition 1 A Scale is a linear table either (1) composed of a centre unit and two sub-level scales, namely, the left sub-level scale and the right sub-level scale or (2) composed of the centre unit only, where

(1) The centre unit positions at the centre of the scale.

(2) The left and right sub-level scales of a scale are scales themselves with equal length.

(3) The scale's centre unit stores the coordinates of the centre units of both its left sub-level scale and its right sub-level scale if it has.

(4) For any scale at any level, only the centre unit stores an item(including operator, variable, constant and coefficient).

A scale is composed of a series of smaller scales at different levels. The lengths of lower level sub-scales scale down(is half, 1/4, 1/8, ⋯, of the current level scale). The scales at the lowermost level are single centre units. All these scales of different levels reside in the highest level scale, level-0 scale, as is shown in Example 2.

Example 2 Fig. 3 shows a sample scale with length L, the centre unit lies in the middle of the scale. Fig. 4 shows scales at subsequent different levels. The scale at level 0 is the scale of the highest level, which is composed of three parts: (1) the centre unit pointed by C_0, (2) the left scale of level 1, starting from 0 to C_0-1 of the highest-level scale, and (3) the right sub-scale, i. e. the right scale of level 1, ranging from C_0+1 to $L-1$ of the highest-level scale.

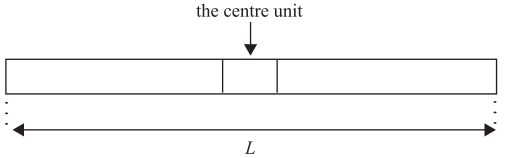


Fig. 3　A Scale with its centre unit

图3　Scale 和中心单元

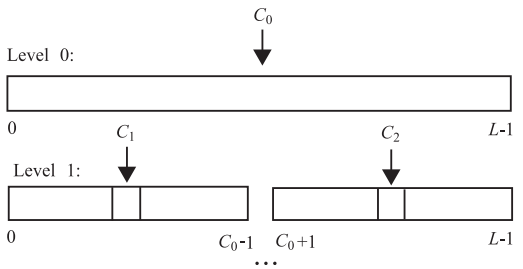


Fig. 4　A Scale and its sub-level Scales

图4　Scale 和亚层子-Scale

In order to efficiently express the gene segments into proper scale places, fast positioning of sub-level centre units is required. Benefited from the careful design of scale, we have Theorem 1.

Theorem 1 Suppose C_n is the centre unit coordinate of a n-level scale, and the length of the scale is L. Then the centre units coordinates of its left and right sub-level scales are $C_n-\lceil L/4\rceil$ and $C_n+\lceil L/4\rceil$, respectively.

Proof (Omitted). □

3.2　The Initialization of Scale

With Theorem 1, the centre units of the left and right sub-level scales can be calculated just from the centre unit position, C_n, when the length of level-0 scale, L, is given. Before gene expression, scale has to be initialized. Algorithm 1 initializes scale.

Algorithm 1 Initialize_Scale

Input: Length of gene head, h

Output: Initialized Scale

Steps:

1. $L=2^h+1$;　　// Set scale length based on h
2. *Scale* = (struct *scale_unit* *) malloc(L * sizeof(struct *scale_unit*));
3. $C_0=\lceil L/2\rceil$　　// Centre unit of level-0 scale
4. *Scale*[C_0].$L=C_0-\lceil L/4\rceil$;　　// Theorem 1
5. *Scale*[C_0].$R=[C_0]+\lceil L/4\rceil$;　　// Theorem 1
6. *anotate*(0, C_0-1, *Scale*[C_0].L);　　// Recursive
7. *anotate*(C_0+1, L, *Scale*[C_0].R);　　// Recursive

void function *anotate*(B_L, B_R, C)

8. $L=B_R-B_L+1$;　　// Calculate sub-level scale length
9. if($L\geqslant 3$){　　// Whether still has sub-level scale

```
10. Scale[C].L = C - ⌈L/4⌉;      // Theorem 1
11. Scale[C].R = C + ⌈L/4⌉;      // Theorem 1
12. anotate(B_L, C-1, Scale[C].L);      // Recursive
13. anotate(C+1, B_R, Scale[C].R);      // Recursive
14. }
```

Problem complexity determines gene head length, h. After h is set, scale length is decided (line 1, Algorithm 1). Scale is initialized to the maximized possible length 2^h+1 (line 2, Algorithm 1). So for any operator in the gene its left and right gene segment operands have been preserved with corresponding places. Then any gene with head length h can be expressed into the same scale with no dynamical construction and release. Therefore, scale is initialized once, keeps static and can be used permanently for all genes to be expressed. Simultaneously, according to Theorem 1, sub-level scale centers are pre-computed and annotated in scale (line 4~7, function *annotate*).

Time consumption of Algorithm 1 is independent of population size, P, generations, G, and number of runs, R. The time cost is trivial (usually below 0.2 seconds).

3.3 Scale-Based Gene Evaluation

This section provides a Naïve gene evaluation approach based on scale. It gives the general skeleton of scale-based gene evaluation.

Algorithm 2 Naïve Evaluation

Input: A Scale with length L; centre point C containing an expressed gene; A fitness instance.

Output: The value of the gene

Steps:

```
1. P = C;      // Set the evaluation entry point
2. return evaluate(P);      // Recursive

double function evaluate(int P) {
3. if(Scale[P].item is an operator Op)      // Deepen
4. return op(Op, evaluate(Scale[P].L), evaluate(Scale[P].R);
       // Recursive
5. else return Scale[P].item;      // Recursion exit
6. }

double function op(char Operator, double L, double R) {
7. switch(Operator) {      // General case 2 operator
8.     case '+':
9.       return(L+R);      // Direct plus
10.    case '-':
11.      return(L-R);      // Direct minus
12.    case '*':
13.      return(L*R);      // Direct multiply
14.    case '/':
15.      return(L/R);      // Direct divide
16.    default:
17.      printf("error in %c\n", Operator);
18.      exit(1);
19.  }
20. }
```

Scale-based gene evaluation starts from the centre unit (line 1~2). If the centre unit contains only a data item, it is returned as the current scale's value (line 5). If it contains an operator, Op, op(left sublevel scale's value, right sub-level scale's value) is returned as the scale's value (line 3 ~ 4, function op). Further, recursive procedure is called to deep-first calculate the left and right sub-level scale's values (function *evaluate*).

4 Gene Evaluation with Variable Matrix

Variable matrix is a matrix containing fitness instances and gene value defined below.

Definition 2 (Variable Matrix) Let N be the number of fitness instances, M be the number of variables in the genes. $A_{N*(M+2)}$ is called a variable matrix if it satisfies the following conditions.

(1) Each column from 1 to M in $A_{N*(M+2)}$ is an assignment vector for the i-th variable of the fitness instances.

(2) The next to last column is the function value (f_j, Fig. 4) vector of the fitness instances.

(3) The last column is the evaluated value (g_i, Fig. 4) vector for the fitness instances.

$A_{N*(M+1)}$ is actually a copy of the fitness instances. The differences of the last two columns are the evaluation errors ($|g_i-f_j|$, Fig. 4) The fitness can be calculated through the last two columns.

With variable matrix single value calculation can be transformed to vector calculation. The fitness instances are no longer looked on as a series of samples that need to be considered one by one, but only a set of vectors that can be used to evaluate genes by only 1 traverse of the scale. With the introduction of variable matrix, Algorithm 2 evolves into Algorithm 3.

Algorithm 3 SEVM

Input: A Scale containing an expressed gene, with length L, centre unit at C;

Variable matrix $A_{N*(M+2)}$;

Fitness function f

Output: The gene's fitness value;

Steps:

```
1. P = C;
2. A_{N*(M+2)}[][M+1] = SEVM_Evaluate(P);
3. return f(A_{N*(M+2)}[][M], A_{N*(M+2)}[][M+1]);

double * function SEVM_Evaluate(int P) {
4. if(Scale[P] is an operator Op)
5.   if(SEVM_Evaluate(Scale[P].L) is a full-value vector,
A_{N*(M+2)}[][m]) {
6.     if(SEVM_Evaluate(Scale[P].R) is a full-value vector,
A_{N*(M+2)}[][n])
7.       for(i=1; i<=N; i++)
       A_{N*(M+2)}[i][m] = op(A_{N*(M+2)}[i][m], A_{N*(M+2)}[i][n]);
8.     else
9.       for(i=1;i<=N;i++)A_{N*(M+2)}[i][m] = op(A_{N*(M+2)}[i][m],
(Scale[P].R)[1]);
10.    return A_{N*(M+2)}[][m];
11. }
12. else if(SEVM_Evaluate(Scale[P].R) is a full-value vector,
A_{N*(M+2)}[][n]) {
13.    for(i=1;i<=N;i++)A_{N*(M+2)}[i][n] = op((Scale[P].L)[1],
A_{N*(M+2)}[i][n]);
```

14.　　return $A_{N*(M+2)}[\,][\,][n]$;
15. }
16. else
17. return $op((Scale[P].L)[1],(Scale[P].R)[1])$;
18. else if($Scale[P]$. item is a variable with reference m)
19.　return $A_{N*(M+2)}[\,][\,][m]$;
20. else return $Scale[P]$;
21. }

Theorem 2 Algorithm 3 has no problem of repetitive traversing and calculation.

Proof Function *SEVM _ Evaluate* recursively traverses the scale (line 5,6,11, Algorithm 3) till the sub-level scale is a variable or coefficient (line 16, 18, Algorithm 3). The function is called only once in gene fitness value calculation (line 3, Algorithm 3). Therefore, algorithm SEVM traverses scale only once. For any level scales computed by two single-value vectors (line 15, Algorithm 3), only one calculation is performed. The calculation among purely single-value vectors is done only once. Therefore, there is no repetitive calculation. □

Example 3 We evaluate the gene in Fig. 2 with variable vectors. As is shown in Fig. 5, the number of calculation operations is 1 000 + 1 = 1 001. The number of node traverses (visit) is 5. Therefore, the total number of operations is 1 006. While in ET-based method, the number is 7 000, almost 7 times as many as scale-based method.

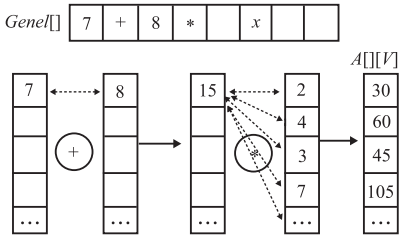


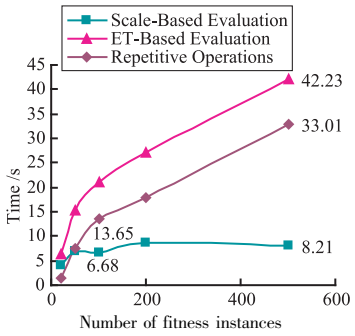
Fig. 5　SEVM Evaluation

图5　SEVM 评估

5　Experiments and Performance Study

This section reports the incipient results on scale-based gene expression and evaluation compared with traditional ET-based methods. All algorithms are implemented with Visual C++ 6.0. on a PC with 2.6 GHz Intel CPU, 256 M memory, running Microsoft Windows XP Professional. The genes are generated randomly by a gene generator function. The user can control the gene head length h, the proportion of operators η, and the proportion of variables ζ. The user can set the population size P, number of generations G, and number of runs R, etc. Since genes are generated randomly, even for the same environments the experiments results may differ to some extent. Therefore, each experiment was conducted 3 times and only the average value was recorded.

Fig. 6 shows the time variances of scale-based evaluation compared with ET-based method. As number of fitness instances increases, ET-based method grows abruptly while scale-based method grows very smoothly. As fitness instances grows from 100 to 500, scale-based evaluation almost keeps steady at 8 seconds,



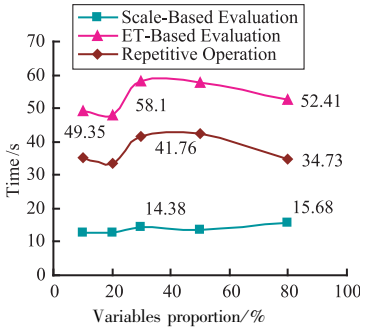
Gene head length, $h = 10$, variable proportion, $\zeta = 20\%$, number of expression, $N = 1$ M

Fig. 6　Effect of fitness instances

图6　适应度实例集的影响

while ET-based evaluation doubles, and the time in repetitive operations grows from 13.65 seconds to 33.01 seconds.

In Fig. 7, we fixed the number of fitness instances at 500, gene head length at 10 and number of expressions at 1 M. ET-based evaluation goes up and down around 55 seconds. When the variable proportion is 80%, scale-based evaluation costs 15.68 seconds, 29% that of ET-based method.

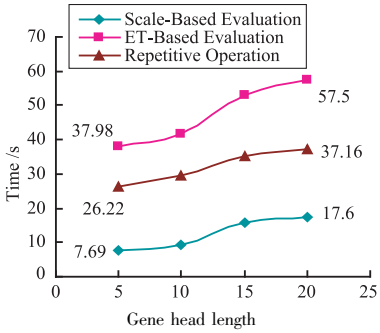


Number of fitness instances, $M = 500$, gene head length, $h = 10$, number of expression, $N = 1$ M

Fig. 7　Effect of proportion of variables ζ

图7　变量比例 ζ 的影响

Fig. 8 displays the experiments results on the increase of gene head length, h. All 3 lines go up as h increases, because larger gene length leads to larger expression. Larger ET results in worse repetitive traverses and calculations, so time consumption of ET-based evaluation grows much faster than that of scale-based evalu-



Number of fitness instances $M = 500$, variable proportion $\zeta = 20\%$, number of expression $N = 1$ M

Fig. 8　Effect of gene head length

图8　基因头长度的影响

ation. When gene head length is 5, scale-based evaluation takes 7.69 seconds, 20% that of ET-based evaluation. As gene head length reaches 20, scale-based evaluation spends 17.6 seconds in the evaluation process, 31% that of ET-based evaluation.

6 Conclusion

Traditional ET-based gene evaluation has a major performance defect, i.e. repetitive traverses and calculations of the ET. This paper proposes a novel model, scale-based GEP with variable matrix. Variable matrix is used to avoid repetitive calculation in gene evaluation. Experiments show scale-based evaluation outperforms ET-based method 3 ~ 5 times constantly.

References:

[1] Ferreira C. Gene expression programming: a new adaptive algorithm for solving problems[J]. Complex Systems, 2001, 13(2): 87-129.

[2] Tang Chang-jie, Li Chuan. Time series prediction based on gene expression programming [C]//LNCS 3129: the 4th International Conference on Web Age Information Mangement 04, 2004: 55-64.

[3] 彭京,唐常杰,李川. M-GEP:基于多层染色体基因表达式编程的遗传进化算法[J]. 计算机学报,2005,28(9):1459-1466.

[4] 陈瑜,唐常杰,叶尚玉,等. 基于基因表达式编程的自动聚类方法[J]. 四川大学学报:工程科学版,2007,39(6):107-112.

[5] Chen Yu, Tang Chang-jie, Zhu Jun, et al. Jiang wu: clustering without prior knowledge based on gene expression programming [C]// Third International Conference on Natural Computation, ICNC, 2007, 3: 451-455.

[6] Davis L. Handbook of genetic algorithms[M]. [S.l.]: Van Nostrand Reinhold, 1991.

[7] Koza J R. Genetic programming: on the programming of computers by means of natural selection[M]. Cambridge, MA: MIT Press, 1992.

[8] Ideker T. A new approach to decoding life: systems biology[J]. Annu Rev Genomics Hum Genet, 2001, 2: 343-372.

[9] Zhou Chi, Nelson P C, Xiao Wei-min, et al. Discovery of classification rules by using gene expression programming [C]// Proceedings of the International Conference on Artificial Intelligence, 2002.

[10] Zuo Jie, Tang Chang-jie, Li Chuan, et al. Time series prediction based on gene expression programming [C]// Advances in Web-Age Information Management, 2004.

[11] Teodorescu L. High energy physics data analysis with gene expression programming [C]// 2005 IEEE Nuclear Science Symposium Conference Record, 2005, 1: 143-147.

[12] Vassilios K. Karakasis and andreas stafylopatis, data mining based on gene expression programming and clonal selection [C]// Proceedings of the IEEE World Congress on Evolutionary Computation, CEC, 2006.

LI Chuan was born in 1977. He received the PhD degree in computer application from Sichuan University in 2006. He is a lecturer at Sichuan University. His research interests include data mining, gene expression programming.
李川(1977-),男,2006 年获得四川大学计算机学院博士学位,目前在四川大学计算机学院做讲师,主要研究领域为数据挖掘,基因表达式编程。

TANG Chang-jie was born in 1946. He received the MSc degree in computational mathematics from Sichuan University in 1982. He is a professor and doctoral supervisor at Sichuan University. His research interests include data mining, gene expression programming.
唐常杰(1946-),男,1982 年获四川大学硕士学位,目前在四川大学计算机学院任教授、博士生导师,主要研究领域为数据挖掘,基因表达式编程。

CHEN Yu was born in 1974. He is a PhD candidate at Sichuan University. His research interests include data mining, gene expression programming.
陈瑜(1974-),男,博士生,主要研究领域为数据挖掘,基因表达式编程。

DAI Shu-cheng was born in 1974. He is a PhD candidate at Sichuan University. His research interests include data mining, gene expression programming.
代术成(1974-),男,博士生,主要研究领域为数据挖掘,基因表达式编程。

QIU Jiang-tao was born in 1972. He is a PhD candidate at Sichuan University. His research interests include data mining, gene expression programming.
邱江涛(1972-),男,博士生,主要研究领域为数据挖掘,基因表达式编程。

LUO Qian was born in 1975. He is a PhD candidate at Sichuan University. His research interests include data mining, gene expression programming.
罗谦(1975-),男,博士生,主要研究领域为数据挖掘,基因表达式编程。

ZHU Jun was born in 1960. She is a professor at Birth Defects Monitoring Center of China.
朱军(1960-),女,中国出生缺陷监测中心教授。

(上接 72 页)

[26] Kim M, Hiroyasu T, Miki M, et al. SPEA2 +: Improving the performance of the strength pareto evolutionary algorithm 2 [C]// LNCS 3242: Proc of PPSN VIII. Berlin: Springer, 2004: 742-751.

[27] Deb K, Agrawal R B. Simulated binary crossover for continuous search space[J]. Complex Systems, 1995, 9(6): 115-148.

[28] Deb K, Beyer H. Self-adaptive genetic algorithms with simulated binary crossover[J]. Evolutionary Computation, 2001, 9(2): 195-219.

[29] Deb K, Goyal M. A combined genetic adaptive search (geneAS) for engineering design[J]. Computer Science and Informatics, 1996, 26(4): 30-45.

[30] Deb K, Mohan M, Mishra S. A fast multi-objective evolutionary algorithm for finding well-spread Pareto-optimal solutions, 2003002[R]. KanGAL, 2003.