文章编号:1001 - 9081(2010)07 - 1896 - 03

新的线性遗传程序设计方法

潘小海,徐蔚鸿,周恺卿
(长沙理工大学 计算机与通信工程学院,长沙 410114)
(jsu_computer@163.com)

摘 要:受其他多种线性编码的遗传程序设计算法的启发,提出一种新的编码方式的遗传程序设计——符号遗传程序设计。该编码方式具有简单、无语法限制并且能够在不增加计算量的情况下将染色体翻译成多个表达式等特点。分析与实验表明该算法具有较高的效率和较强的稳定性。

关键词:遗传程序设计;进化算法;符号回归;线性编码
中图分类号: TP18 **文献标志码**:A

New linear genetic programming approach

PAN Xiao-hai, XU Wei-hong, ZHOU Kai-qing
(*School of Computer and Communication Engineering, Changsha University of Science and Technology, Changsha Hunan* 410114, *China*)

Abstract: A new genetic programming named Symbol Genetic Programming (SGP) based on a new encoding method was proposed. This new encoding method absorbed the merits of many other linear genetic programming methods. It coded with a simple, unrestrained string. Based on its characteristic, multi-expressions could be contained in one individual without the increase of computation task. This method is proved to be effective and stable through the complexity analysis and experiment.

Key words: Genetic Programming (GP); evolutionary algorithm; symbolic regression; linear representation

0 引言

遗传程序设计(Genetic Programming, GP)[1]是进化技术的一种,用于自动生成计算机程序,传统 GP(Tree Based GP)的程序个体采用非线性结构编码,用树结构表示,但存在处理树结构效率较低,可能引起代码膨胀,影响搜索,且算法实现较困难等问题。各国学者在 GP 的基础上相继提出线性结构编码和图结构编码等方案,其中的线性结构编码 GP 迅速成为研究热点,具有代表性的有 GE (Grammatical Evaluation)[2-3]、LGP (Linear Genetic Programming)[4]、GEP (Gene Expression Programming)[5]、MEP (Multi Expression Programming)[6-7]等,这些算法都具有将基因型和表现型分离的特性,个体被编码成一个线性实体,并能转换成非线性结构。

本文在已有文献的基础上,受线性编码与多表达式编码的启发,设计了没有任何语法限制的多表达式编码方式,并描述了编码的特性和表达能力,在此基础上提出了基于此编码的符号遗传程序设计(Symbol Genetic Programming, SGP)算法。

1 个体表示

1.1 个体的线性编码

逆波兰表示法(Reverse Polish Notation, RPN)把所有操作符置于操作数的后面,因此也被称为后缀表示法,称逆波兰表示的表达式为后缀表达式,它由操作符和终结符组成,如果操作符的元数是固定的,则语法上不需要括号仍然能被无歧义地解析,因此后缀表达式中不需要括号描述。例如后缀表达式((x y +) (a b *) /),去掉括号成为(x y + a b * /),对应的数学表达式是$(x+y)/(a*b)$,其中(x, y, a, b)是终结符,(+, /, *)是操作符。

后缀表达式的解释器一般是基于堆栈的,解释过程如下:从头至尾扫描表达式中的符号,遇操作数就压入堆栈,遇操作符就从堆栈弹出操作数并求值,将所得值入栈。所有符号处理完毕,栈顶元素就是表达式的值。故后缀表达式是一种方便和性能优良的表达工具,能够很简便地进行解释求值。

后缀表达式是操作符和终结符组成的符号序列,设操作符集合为 F,终结符集合为 T,则符号集为 $S = F \cup T$,但并非所有的符号序列 $s = (s_1 s_2 s_3 \cdots s_k)$, $s_i \in S$ 均为合法后缀表达式,例如序列(x - ya *)不是合法的后缀表达式。

为了使任意符号序列 $s = (s_1 s_2 s_3 \cdots s_k)$, $s_i \in S$ 都能转换成后缀表达式或者数学表达式,提出算法 1,具体描述如下。

算法 1 符号串到后缀表达式/数学表达式集合的转换。

输入:符号序列 $L = (s_1 s_2 s_3 \cdots s_k)$,操作符集合 F,终结符集合 T;

输出:集合 *output*,数组 e。

```
for each s in S
  if ( s 是操作符)
    if ( 堆栈长度少于操作符的元数 n)
      n 个元素弹出堆栈;
      令 e[i] = 出栈元素和操作符组成的后缀表达式/数学表达式;
      e[i] 入栈并加入 output 集合;
      i = i + 1;
    end if
    else if( s 是终结符)
```

收稿日期:2010 - 01 - 04;**修回日期**:2010 - 03 - 03。
基金项目:教育部重点科研基金资助项目(208098);湖南省教育厅重点项目(07A056)。
作者简介:潘小海(1982 -),男,湖南武冈人,硕士研究生,主要研究方向:智能软件系统; 徐蔚鸿(1963 -),男,湖南湘潭人,教授,博士生导师,博士,主要研究方向:智能系统、模式识别、软件工程; 周恺卿(1984 -),男,湖南长沙人,硕士研究生,主要研究方向:Petri 网、计算智能。

```
        令 e[i] = s;
        e[i] 入栈并加入 output 集合;
        i = i + 1;
      end if
    end for
```

设有符号序列 $s = (--zy + y * /xz - * / + /)$，通过算法 1 转换成数学表达式集合是 $\{z, y, (z + y), ((z + y) * y), x, (x - z), (((z + y) * y) * (x - z))\}$，转换成后缀表达式集合是 $\{(z), (y), (zy +), (zy + y *), (x), (xz -), (zy + y * xz - *)\}$，算法中的 $e[i]$ 表示符号串中蕴含的第 i 个表达式。

1.2 表达能力分析

由算法 1 确定了一个转换 F:符号序列 → 后缀表达式集合，并且 F 满足下面的定理。

定理 1　$F(x) \subseteq F(xy)$，其中 x 和 y 是任意符号序列。

证明　设 $x = (x_1 x_2 \cdots x_m)$，$y = (y_1 y_2 \cdots y_n)$，$s = xy = (x_1 x_2 \cdots x_m y_1 y_2 \cdots y_n)$，在对 s 的转换过程中，处理完符号串 $x_1 x_2 \cdots x_m$ 之后，输出集合为 o_1，于是 $F(x) = o_1$，当处理完符号 $y_1 y_2 \cdots y_n$ 后，输出集合为 o_2，于是 $F(xy) = o_2$，由于 o_2 是在 o_1 中加入了若干个元素，所以 $o_1 \subseteq o_2$，于是 $F(x) \subseteq F(xy)$。

定理 2　$e \subseteq F(e) \subseteq F(xey)$，其中 e 是后缀表达式，x 和 y 是任意符号序列。

证明　设 $x = (x_1 x_2 \cdots x_m)$，$e = (e_1 e_2 \cdots e_n)$，$s = xe = (x_1 x_2 \cdots x_m e_1 e_2 \cdots e_n)$，在对 s 的转换过程中，当处理完符号 $x_1 x_2 \cdots x_m$ 时，输出集合为 o_1，堆栈中的元素为 $(p_1 p_2 \cdots p_k)$，p_k 是栈顶元素，当处理 $e_1 e_2 \cdots e_n$ 的过程中，由于 $e_1 e_2 \cdots e_n$ 是合法后缀表达式，因此转换过程与堆栈中已有元素无关，即不会出现从堆栈中弹出 p_k 的情况，设在这个过程中往 o_1 中加入的元素是 $\{t_1, t_2, \cdots, t_r\}$，其中 $t_r = e$，所以 $F(e) = \{t_1, t_2, \cdots, t_r\}$，同时 $F(xe) = o_1 \cup \{t_1, t_2, \cdots, t_r\}$，可得 $e \subseteq F(e) \subseteq F(xe)$，由定理 1 知 $F(xe) \subseteq F(xey)$，所以 $e \subseteq F(e) \subseteq F(xey)$。

长度为 k 的符号序列构成的空间表示为 $C = S^k = \{(s_1 s_2 s_3 \cdots s_k) \mid s_i \in S\}$，长度不大于 k 的后缀表达式的集合(包括空表达式)组成空间 E，元组 (C, E, F) 具有以下特性:

性质 1　$e \in E$，$|e| = l$，满足:

$$|\{c \mid c \in C \wedge e \in F(c)\}| \geqslant (k - l + 1)|S|^{(k-l)}$$

证明　由定理 2 可得 $e \subseteq F(xey)$，集合 $\{xey \mid xey$ 的长度等于 $k\}$ 中的元素数目为 $(k - l + 1)|S|^{(k-l)}$，所以等式成立。

性质 2　$\bigcup_{c \in C} F(c) = E$。

证明　任意 $e \in E$，$|e| = l \leqslant k$，由性质 1 可知，必然存在 $(k - l + 1)|S|^{(k-l)} \geqslant 1$ 个符号序列经 F 转换之后包含 e，同样任意 $c \in C$，$F(c) \subset E$，所以 $\bigcup_{c \in C} F(c) = E$ 成立。

对于元组 (C, E, F)，可以把空间 C 看做是编码空间(基因型空间)，E 看作是问题空间(表现型空间)，F 看作是编码空间到问题空间的映射。由性质 1 和性质 2 可知，编码是完备的。即任意表达式都至少存在一个编码且任意编码都可以表示一组表达式。

2 SGP 算法流程

2.1 个体的解释与评价

个体的解释方法与算法 1 大致相同，不同的是，遇到终结符时，入堆栈的是代表终结符的值，遇到操作符时入堆栈的是计算的结果，而不是表达式。

将个体的解释算法看成一个函数 $P(e, \boldsymbol{L})$，e 表示个体，$\boldsymbol{L}$ 表示终结符值列表，那么函数 P 的返回值是一个向量，对同一个个体 e，$P(e, \boldsymbol{L})$ 的维数是一样的，令 $P_i(e, \boldsymbol{L})$ 表示第 i 个分量，则 $P_i(e, \boldsymbol{L})$ 即是个体 e 中第 i 个表达式的值。并且可以得出，解释个体的时空复杂度与编码长度呈线性关系。

2.2 算法流程

假设要解决的问题表示为:

矩阵 $\boldsymbol{D}_{m \times n} = \begin{bmatrix} \boldsymbol{L}_1 & O_1 \\ \boldsymbol{L}_2 & O_2 \\ \vdots & \vdots \\ \boldsymbol{L}_n & O_n \end{bmatrix}$，$\boldsymbol{L}_j$ 是行向量，O_j 是 $\boldsymbol{L}_j$ 对应的目标值，求函数 P，使 $\sum_j |P(\boldsymbol{L}_j) - O_j|$ 最小化。

个体 e 中第 i 个表达式的原始适应度表示为 $R_i(e) = \sum_j |P_i(e, \boldsymbol{L}_j) - O_j|$，个体的原始适应度为所有表达式中原始适应度的最小值，即 $R(e) = \min_i R_i(e)$。

要解决的问题转换为:寻找 $e \in C$，使 $R(e)$ 最小化，因此可用类似于遗传算法(Genetic Algorithm，GA)和 GP 算法的框架，来求解此类问题。简要流程如算法 2。

算法 2　SGP 框架。

```
产生初始种群 P;
while(结束条件不成立)
  评价种群 P,保存最好个体到 best;
  for i = 1 to PopuliationSize/2
    从 P 中选出 2 个体复制到 p1,p2;
    if ( rand() < Pc)
      p1 和 p2 进行一点或两点交叉;
    end if
    p1 和 p2 进行变异;
    p1 和 p2 加入到临时种群 P';
  end for
  令 P = P';
  最好个体 best 替换 P 中随机选定个体;
  临时种群 P' 设置为空;
end while
输出最好个体
```

2.3 个体遗传操作

2.3.1 初始个体的产生

初始个体的产生有很多方式，这里采用比较简单的一种，初始个体中的每一位符号有 $a(0 < a < 1)$ 的几率从终结符集合中随机取得，$1 - a$ 的几率从终结符集合中随机取得，a 的值决定了初始个体的分布，它可以根据操作符集合中的函数的平均元数来决定，在本文的实验中均采用 $a = 0.5$。

2.3.2 遗传操作

设定个体变异操作的概率为 1，当个体执行变异操作时，个体的每一位都有概率 Pm(位变异率)变成随机符号，随机符号的选取与初始个体的产生中随机符号的选取方式相同。

选出的两个父代个体进行交叉操作的概率称为交叉概率(Pc)，使用的交叉方式是基于遗传算法中的一点交叉和两点交叉。进行一点交叉和两点交叉的概率分别称为一点交叉率 $P1r$ 和两点交叉率 $P2r$，并且满足 $P1r + P2r = 1$。

3 实验仿真

本文使用两个实验分别测试编码方式的特性和 SGP 算

法的性能。实验结果表明 SGP 算法有较高的成功率、比较少的进化代数以及运行的稳定性。

3.1 确定初始个体中表达式最大长度的分布。

使用操作符集 $F = \{+, -, *, /\}$，终结符集 $T = \{x, y\}$，使用上文所述的初始个体产生算法，取参数 $a = 0.5$，编码长度取自序列(10, 15, 20, …, 400)，每一点产生200个初始个体 $e_i(1 \leqslant i \leqslant 200)$，计算个体 e_i 蕴含的最长表达式的平均值 $A = \frac{1}{200}\sum_{i=1}^{200}(\mathrm{Max}(F(e_i)))$，$\mathrm{Max}(F(e_i))$ 表示集合 $F(e_i)$ 中表达式的长度最大值，实验结果如图1、图2所示。

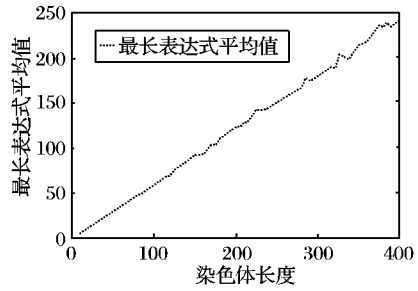

图1 最长表达式的平均值与染色体长度的关系

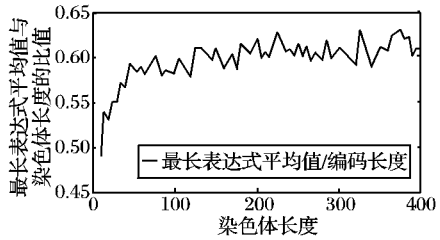

图2 最长表达式的平均值与染色体长度的比率关系

实验表明，最长表达式的长度平均值随染色体长度线性增大，并且它与编码长度的比率 v 大部分位于区间[0.5,0.65]。实验初步表明比率 v 不受操作符集和终结符中元素的个数影响，但是受参数 a 和操作符集合中操作符的元数影响。

3.2 符号回归对比实验

这里采用的测试函数如下：

$$f(x) = 4x^4 + 3x^3 + 2x^2 + x + 1$$

测试数据集从[-5,5]平均取20个点。为测试算法的各种性能，与符号回归中表现较好的基因表达式程序设计(Gene Expression Programming，GEP)算法进行对比实验，实验中使用的参数见表1，实验结果见图3。

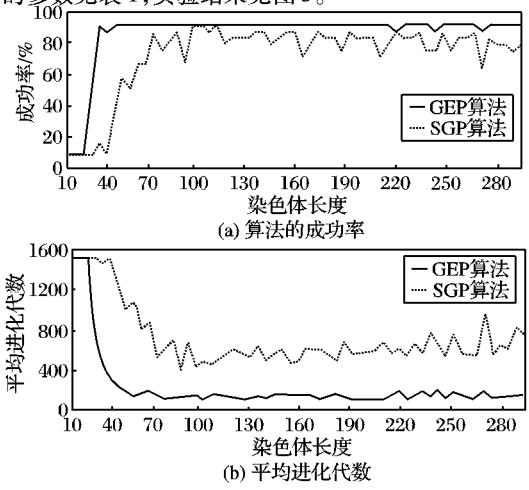

图3 成功率及平均进化代数曲线

从图3可以看出，SGP测试中，染色体长度大于40时，成功率基本为1，平均进化代数稳定在150左右，并不随染色体长度变化而变化。GEP算法成功率较低，染色体长度大于70时，成功率0.85左右，平均进化代数基本大于600，并随染色体长度增大有增加的趋势。

表1 实验参数

参数	GEP算法取值	SGP算法取值
运行次数	50	50
群体规模	100	100
代数限制	1500	1500
位变异率 P_m	0.03	0.03
交叉概率 P_c	—	0.7
一点交叉率 P_{1r}	0.3	0.5
两点交叉率 P_{2r}	0.3	0.5
基因交叉率	0.3	—
转座率	0.1,0,1,0.1	—
转座元素	1,2,3	—
基因数量	1	—
选择方法	2-Tournament selection	2-Tournament selection
操作符集	{+,-,*,/}	{+,-,*,/}
终结符集	{x, 1.0}	{x, 1.0}
终止条件	绝对误差<10e-5	绝对误差<10e-5
染色体长度	(10,15,20,…,300)	(10,15, 20,…,300)

4 结语

受线性编码与多表达式编码的启发，提出了没有任何语法限制的多表达式编码方式，描述了编码的特性和表达能力，提出了基于此编码的SGP算法。并且在符号回归的实验中与GEP算法做了对比，证明SGP算法有较高的成功率比较少的进化代数运行稳定。

参考文献：

[1] KOZA J R. Genetic Programming I[M]. Cambridge: MIT Press, 1992.

[2] O'NEILL M, RYAN C. Grammatical evolution[J]. Evolutionary Computation, 2001,5(4): 349-358.

[3] BURBIDGE R, WALKER J H, WILSON M S. Grammatical evolution of a robot controller[C]// Proceedings of the 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems. Piscataway, NJ: IEEE, 2009: 357-362.

[4] BRAMEIER M, BANZHAF W. A comparison of linear genetic programming and neural networks in medical data mining[J]. IEEE Transactions on Evolutionary Computation, 2001, 5(1): 17-26.

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

[6] OLTEAN M, GROSAN C. Evolving digital circuits using multi expression programming [C]// 2004 NASA/DoD Conference on Evolvable Hardware. Washington, DC: IEEE Computer Science, 2004: 87-94.

[7] CHEN YUEHUI, JIA GUANGFENG, XIU LIMING. Design of flexible neural trees using multi expression programming[C]// Proceedings of Chinese Control and Decision Conference. New York: IEEE, 2008: 1429-1434.