国防科技大学学报
JOURNAL OF NATIONAL UNIVERSITY OF DEFENSE TECHNOLOGY
第18卷第4期 1996年12月 Vol. 18 No. 4

程序进化器*

吴少岩 陈火旺

(国防科技大学计算机系 长沙 410073)

摘 要 研究用模拟进化方法进行程序设计的途径。构造了以形式文法为遗传表示的程序进化器，提出了这个进化器使用的有关算法。本文用程序进化器解决了人工蚁问题。

关键词 进化计算，程序设计，搜索

分类号 TP311

Program Evolver

Wu Shaoyan Chen Huowang

(Department of Computer Science, NUDT, Changsha, 410073)

Abstract A programming approach by means of simulated evolution is investigated. A program evolver which uses a formal grammar as its genetic representaion is constructed and the corresponding algorithms in the evolver are presented. Finally, the evolver is applied to the artificial ant problem.

Key words evolutionary computation, programming, search

自然进化是基于群体的随机优化过程，用机器模拟这一过程可产生有效的优化技术，由于模拟进化方法极强的求解问题能力，因而被实际应用于工程优化、自适应控制、预测、机器学习等等。

程序设计是复杂的智能行为，“不用明确编程，而让机器学会解题”是计算机科学的核心问题之一[1]。最近，用模拟进化方法辅助程序设计的研究已经取得重要进展。Koza将常规遗传算法[2]框架直接推广至程序空间，即用LISP的S表达式替换二进制串作为遗传表示[1,3]。Koza基于实验的工作表明：对许多问题，通过遗传程序设计，机器能自动地构造出解决问题的程序。

本文基于一种新的进化计算模型[1,5]，以形式文法为遗传表示构造一个用于搜索程序空间的程序进化器，提出有关算法，并以人工蚁问题为例说明这个进化器的寻优能力。

* 国家863高技术基金资助
1996年5月9日收稿。

1 表示模式与遗传算子

传统的进化算法采用固定、无层次的简单结构作为搜索空间中每个解的编码表示。程序进化器搜索程序空间，需要维持一个程序群体的进化，而程序的结构具有层次性和形状多样性。对某个具体任务的程序空间而言，其最优程序的大小和形状也是预先未知的。

考虑下述形式文法

G：

$G=\{V_N, V_T, S, P\}$

$V_N=\{S, F, E, A\}$

$V_T=\{f_1^{(n_1)}, \cdots, f_m^{(n_m)}, v_1, \cdots, v_k, c_1, \cdots, c_d, [,]\}$

P：

①$S\rightarrow[F]|E$

②$F\rightarrow f_1^{(n_1)}A^{n_1}|\cdots|f_m^{(n_m)}A^{n_m}$

③$A\rightarrow S$

①$E\rightarrow v_1|\cdots|v_k|c_1|\cdots|c_d$

其中 A^n 表示 n 个 A 的并置。

由文法 G 生成的语言 $L(G)$包含具有层次性、长度大于零的各种句子。这个文法为程序编码提供了一个抽象模板。如果将终结符'['，']'解释为通常意义下的括号，则这两个符号表示了个体结构中的层次性。终结符 $f_i^{(n_i)}$ 可解释为 n_i 元算子($n_i>0$)，而 v_i 则解释为变元，c_i 解释为常元。当对上述形式文法的终结符赋予特定含义之后，语言 $L(G)$便代表了一个特定的程序空间。

如果将 $L(G)$的句子看成是具有层次的、可变长的基因串(染色体)，我们便可以讨论染色体上发生的遗传操作。

设串 r，s 为 $L(G)$的句子，$m=|r|$，$n=|s|$分别为 r，s 的长度，交配概率为 P_c，并设过程 sort(t)把终结符分为四个类别：

(1)f:$f_1^{(n_1)}, \cdots, f_m^{(n_m)}$；　(2)$e$:$v_1, \cdots, v_k, c_1, \cdots, c_d$；　(3)'['：[；(4)']'：].

程序进化器采用一种新的进化计算模型 PEBE[4, 5]，其交配算子 O_c 要求从双亲中优化地产生 $2d$ 个新子女($d>1$)。不失一般性，设句子 r 的适应值不次于 s。下述算法 Syntax_Oc 是应用于句子 r 和 s 的交配算子，它于 r 上产生 d 个不同的交配点。于 s 上产生单个交配点。r 的 d 个交配点确定的 d 个子句同 s 的单个交配点确定的一个子句依次交换，产生 $2d$ 个不同的句子作为 O_c 算子的交配结果。这种交配方式对较好的个体用另一个体的部分结构作多次扰动，期望发现进一步改进较好个体的结构组合，或者提出较好个体的部分结构来改进较差的个体。

算法 Syntax_ Oc：作用于句子 r 和 s 的 O_c 算子，产生 $2d$ 个子女。

Syntax Oc(r, s, d)

{　生成随机实数 $w\sim U[0, 1]$；

　　if $w\leqslant P_c$ 顺序完成下述 i)～iii)；

i)生成 d 个不同的随机整数 $k_1, \cdots, k_d\sim U[1, m]$及随机整数 $k\sim[1, n]$；

ii)While(sort($s[k]$)$\neq e$ and sort($s[k]$)$\neq$ '[') $k=k-1$;

iii)for($i=1$ to d)

{while(sort($r[k_i]$)$\neq e$ and sotr($r[k_i]$)$\neq$ '[')$k_i=k_i-1$;

确定交配段：

if sort($r[k_i]$)$=e$ then 复制 e 至 r 的交配段 segr；

else

{确定与 $r[k_i]$即'['配对的']'，设为 $r[k_i+h_i-1]$；

复制 $r[k_i]$，…，$r[k_i+h_i-1]$至交配段 segr；

}

类似地，

if sort($s[k]$)$=e$ then 复制 e 至 s 的交配段 segs；

else

{确定与 $s[k]$即'['配对的']'，设为 $s[k+h-1]$；

复制 $s[k]$，…，$s[k+h-1]$至交配段 segs；

}.

交配段互换，产生两个新串 u_i，v_i：

$u_i=r[1]\cdots r[k_i-1]//\text{segs}//r[k_i+l_i]\cdots r[m]$；

$v_i=s[1]\cdots s[k-1]//\text{segr}//s[k+h]\cdots s[n]$；

2 初始群体的生成

现在我们考虑程序进化器如何从语言 $L(G)$中采样 $2M$ 个句子作为初始群体。先定义有关概念，文法 G 的定义同前。

定义(子句)

设 $r\in L(G)$，符号串 s 称为句子 r 的子句，如果 s 是 r 的子串且 $s\in L(G)$。

定义(f 集、e 集和 t 集)

我们给文法 G 的终结符分类并作不同记号：称集合$\{f_1^{n_1},\cdots,f_M^{n_M}\}$为 G 的 f 集；称$\{v_1,\cdots,v_k,c_1,\cdots,c_d\}$为 G 的 e 集；称$\{f_1^{n_1},\cdots,f_M^{n_M}\}\cup\{v_1,\cdots,v_k,c_1,\cdots,c_d\}$为 G 的 t 集；

定义(嵌套层)

对一个句子中包含的每个 t 集符号标记一个自然数 c，该自然数称为被标记符号的嵌套层。标记规则如下：

对任意 $s\in L(G)$，i)如果 s 为 e 集单个符号，则该符号的嵌套层标记为 0。ii)如果 s 为$[f_i^{(n_i)}\cdots]$形式，则标记 $f_i^{(n_i)}$ 的嵌套层为 0。iii)设$[f_j^{(n_j)}a_1\cdots a_{n_j}]$为 s 的子句且 $f_j^{(n_j)}$ 已被标记为 c，则：若 $a_i(i=1,\cdots,n_j)$为 e 集符号，则 a_i 被标记为 $c+1$；否则，若 a_i 为$[f_k^{(n_k)}\cdots]$，$f_k^{(n_k)}$ 也被标记为 $c+1$。

定义(最大嵌套层)

当一个句子的所有 t 集符号全被标记时，将嵌套层的最大值称为句子的最大嵌套

层，简称句子的嵌套层。

嵌套层的概念在一定程序上表明了句子结构的复杂性。

一般地，语言 $L(G)$ 是无穷集，即 $L(G)$ 包含无穷多个句子。这也说明程序的搜索空间是无穷的(对比：遗传算法的搜索空间为有限集)。初始群体要求包含搜索空间的有限采样。事实上，我们无法使初始群体包含任意复杂的句子。在对最优句子(最优程序)的结构、形状没有先验知识的情况下，需依据两条基本准则产生初始群体：①在群体中，e 集各个符号有等出现机会；②群体包含的句子具有结构多样性。

下述第一种算法产生最大嵌套层为 c 的句子；第二种算法产生的句子，其最大嵌套层小于 c。设 T 为算法输出的句子。

算法 Gen_Stl：生成最大嵌套层为 c 的句子。

```
Gen_Stl(c)
{if c=0 then 随机从 e 集中取符号 x;T←x;返回 T;
  else{T←'['; 随机从 f 集中取符号 f_i^(n_i);
    for(j=1 to n_i)
    T←T//Gen_Stl(c−1);
    T←T//']';
    返回 T; }
  }
```

算法 Gen_St2：生成最大嵌套不大于 c 的句子。

```
Gen_St2(c)
{if c=0 then 随机从 e 集中取符号 x;T←x;返回 T;
  else{随机从 t 集中取符号 y;
        if y∈e 集 then{T←y;返回 T; }
        else{y∈f 集，设 y 为 f_i^(n_i)，则 T←'[';
            for (j=1 to n_i)
              T←T//Gen_St2(c−1);
            T←T//']'; }
        返回 T; }
}}
```

初始群体的生成是通过反复使用上述两个算法产生的 $2M$ 个句子。下述算法生成初始群体，其中 C 为允许的嵌套层上限。

算法 Gen_Ini_Pop：随机生成 $2M$ 个句子组成初始群体。

```
Gen_Ini_Pop(C)
{for (i=1 to m)
      {生成两个随机整数 c1, c2~U[0, C];
        调用 Gen_Stl(c1)产生第 2i−1 句子;
        调用 Gen_St2(c2)产生第 2i 句子; }
}
```

3 适应值的计算

适应值用来衡量群体中个体的相对优势，是选择的基本依据。没有选择，进化便不会产生。适应值的计算依赖程序设计面对的具体任务，它因程序空间的不同而有所不同。

文法 G 产生的句子是一个抽象的符号串，只有当每个符号都赋予了具体含义时，整个句子才有意义。一般地，我们用三元组$<PO, \nu, c>$代表一个程序空间。PO 为算子集，ν 为算子集，而 c 为常元集。程序空间中的程序可解释文法 G 的句子，让 PO 算子集中每个 n_i 元算子 $f_i^{n_i}$ 对应 f 集中的符号 $f_i^{(n_i)}$，用变元集中 v_i 对应语法符号 v_i，常元集中 c_i 对应语法符号 c_i。于是，由文法 G 产生的任何句子都变为有语义规定的程序。

因此，群体中的一个句子在计算适应值需经历以下步骤：

①句子中的每个语法符号赋予语义解释(binding)，使句子成为程序。

②利用量度方法，获得程序的量度值。

③将量度值做适应变换，获得句子的适应值。

事实上，上述三个步骤未必是决然分开的。步骤①、②可合并在一起进行，其方法是：保留程序空间与 $L(G)$ 之间的一个映射，在句子的语法符号的控制下，逐一解释每个符号，一旦形成可计值子句便计算其值。如果第②步计算出的量度值本身可做为适应值，则适应变换步骤③可以省去。

设程序空间$\langle PO, \nu, c\rangle$，给定变元集中各变元一组输入值，$s$ 是一个句子。算法 Eval(s)完成上述前两步，函数 Bind(x)完成语法符号 x 的语义解释和(或)赋值。

算法 Eval：计算句子的量度值。

```
Eval(s)
{if s 是 e 集单个符号 x then {val←Bind(x);返回 val;}
    else /* s 必为符号串[f_i^(n_i) a_1…a_n_i]* /
    {f_i^n_i←Bind(f_i^(n_i));/* 把 f_i^(n_i) 解释为 n_i 元初等算子* /
      for(j=1 to n_i)
        val_j←Eval(a_j);
      val←f_i^n_i(val_1, …, val_n_i);返回 val; }
}
```

如果适应变换为 $f: R\to R^+$，则群体中个体 s 的适应值为 $f(\mathrm{Eval}(s))$.

4 程序进化器的迭代过程

利用前述算法，基于 PEBE 模型的程序进化器可归纳为下述迭代过程：

设 Ω 为程序空间，Eval：$\Omega\to R$，Syntax_Oc：$\Omega^2\to\Omega^2$，SMUT：$\Omega\to\Omega$，T 为迭代界限。(其中：SMUT 为简单变异算子)。

(1)$t=0$；用 Gen_Ini_Pop(C)产生初始群体 $B_t=\{x_1, \cdots, x_{2M}\}$；

(2)$\forall\, x_i\in B_t$，计算 Eval(x_i)，$i=1, \cdots, 2M$。保留 B_t 中“最好”x_i 于 X^0；

(3)令 $p_i=\dfrac{f(\mathrm{Eval}(x_i))}{\sum_{j=1}^{2M} f(\mathrm{Eval}(x_j))}$，$i=1, \cdots, 2M$。依概率 p_i 用“转轮法”从 B_t 中选 $2M$ 个解

放入 B_t'，$B_t'=\{x_1', \cdots, x_{2M}'\}$；其中 $f:R\to[0, \infty)$。

(4)$\forall x_{2i-1}', x_{2i}' \in B_t'$，$\{i=1, \cdots, M\}$，生成 $r_i \sim U[0, 1]$，若 $r_i > P_c$ 复制 x_{2i-1}'，x_{2i}'到 B_t''；否则

(a)计算 Syntax_O$_c(x_{2i-1}', x_{2i}', d)=(c_1', \cdots, c_{2d}')$；

(b)计算 Eval(c_j')，$(j=1, \cdots, 2d)$；

(c)从 $c_1', \cdots, c_{2d}'$中选取两个“较好”者放入 B_t''；；

(5)$\forall x_i'' \in B_t''(i=1, \cdots, 2m)$，

(a)依概率 P_m' 计算 SMUT$(x_i'')=x_i$；若 SMUT 发生，则计算 Eval(x_i)；

(b)将 x_i 放入 B_t；

(6)若 B_t 中“最好”者优于 X^0，则更新 X^0；

(7)$t=t+1$；若 X^0 为满意解或 t 已达到 T，终止；否则，继续(3)。

5 人工蚁问题

人工蚁是著名的机器人规划(Robotic Planning)问题[1]。一个人工蚁被放置在正方形网格平面上。平面上不规则地摆放了一些食物。目标是为人工蚁设计一个导航程序，使其在一定时间限制下，找出平面上的所有食物。“Santa Fe 轨迹”是该问题的一个具体案例：网格平面大小为 32×32 个方格，在其上放置了 89 块食物，人工蚁的出发点在坐标(0, 0)(左上角)且方向朝东。食物在平面上不是连续放置的，可能有 1～3 个方格的间隔，且存在拐角，如图 1。图中黑方块代表食物。人工蚁的视野很有限，它只能看到当前面对的下一个方格，所能完成的动作包括下述几种：

- RIGHT 右转 90°(不移动)
- LEFT 左转 90°(不移动)
- MOVE 沿当前方向移动一方格，若移动后的方格中有食物，则吃掉食物。方向不变。

人工蚁寻找食物所花的时间，可用它完成的动作数目来衡量。设预先给定的动作数目上限为 AU，网格平面上的食物总数为 FU，则一个人工蚁导航程序要达到的目标 H 可定义为：

H：(aN$\leqslant$AU)$\wedge$(fN=FU)或记为：fN=FU s.t. aN$\leqslant$AU。

这里，aN 和 fN 分别表示在一个导航程序的引导下，人工蚁实际完成的动作数目和发现的食物总数。

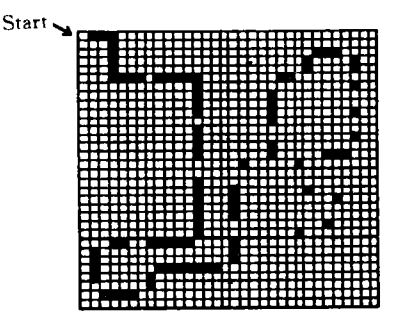

图 1 人工蚁问题的实例：Santa Fe 轨迹

当利用程序进化器发现这一程序时，首先我们要确定该程序所属的程序空间。常元集包含人工蚁的三个基本动作：左转(“LEFT”)，右转(“RIGHT”)和移动(“MOVE”)；变元集为空。程序空间的算子集有三个简单的自定义算子：IF_FOOD，P2，P3。算子 IF_FOOD 带

有两个参数，如果人工蚁面临的前一位置有食物，则计值参数1，否则计值参数2；P2和P3分别带有两个和三个参数，它们都是代表顺序计值参数的算子。量度程序性能的方法是用程序模拟导航人工蚁的活动，在一定时间范围限定下，记录人工蚁寻找到的食物数目。我们将人工蚁允许完成的最大操作数目作为程序的时间约束，人工蚁的一次转向或一次移动都计做一次有效操作，操作计数器增一。在满足时间约束的前提下，能引导人工蚁找到所有食物的程序，即为最优的程序。

程序进化器计算程序空间中169348个点，于第32代给出的下述程序，能在400个操作的约束下寻找到Santa Fe轨迹上的所有89块食物：

[IF_FOOD MOVE [P3[IF_FOOD[P3 RIGHT MOVE RIGHT]RIGHT][P3[IF_FOOD MOVE RIGHT]MOVE[IF_FOOD[P3[IF_FOOD[P2 MOVE MOVE]RIGHT] MOVE RIGHT][IF_FOOD MOVE RIGHT]]][IF_FOOD[P2[P3 MOVE MOVE RIGHT]RIGHT]RIGHT]]]

值得指出，Koza使用遗传程序设计方法给出的人工蚁问题的另一程序[1]：

[IF_FOOD MOVE[P3[LEFT[P2[IF_FOOD MOVE RIGHT][P2 RIGHT [P2 LEFT RIGHT]]]][P2[IF_FOOD MOVE LEFT]MOVE]]

不能使人工蚁在400个操作内寻找到所有89块食物。验证表明Koza给出的程序需要人工蚁完成545个操作才能达到目的。

6 结 论

本文基于新的进化计算模型构造了一个程序进化器，并以此解决了人工蚁问题。将模拟进化方法应用于程序设计是一个重要的研究方向，为自动程序设计开辟一条新途径。

参考文献

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

2 Goldberg D E. Genetic algorithms in search, optimization and machine learning. Addison Welsey, Reading, MA, 1989

3 Koza J R. Genetic programming Ⅱ: automatic discovery of reusable programs. Cambridge, MA: MIT Press, 1994

4 Wu S Y, Zhang Q F, Chen H W. A new evolutionary model based on family eugenics:the first results. IEEE 3rd international conference on evolutionary computation, Japan, 1996.

5 吴少岩．基于遗传与进化原理的搜索、优化技术:[博士学位论文]. 长沙：国防科技大学计算机系，1996，4

（责任编辑 张 静）