基于自适应遗传算法的软件测试用例自动生成①

李 柱

(重庆交通大学, 重庆 400074)

摘 要: 在软件测试中, 测试成功的关键是快速、高效的生成测试用例. 遗传算法是一种通过模拟自然界生物进化过程搜寻最优解的一种算法, 算法通过选择、交叉和变异操作引导算法搜索方向, 逐步接近全局最优解. 传统遗传算法由于具有较好的全局搜索能力, 因此被很多科研人员应用于测试用例生成. 但遗传算法的固有缺陷“早熟收敛”, 容易导致算法收敛于局部最优. 针对这种情况, 提出一种自适应遗传算法, 该算法交叉算子和变异算子可根据程序变化自动调整, 随后, 将改进后的算法应用于一程序的测试用例生成中. 测试结果表明该算法在测试用例生成的效率和效果方面优于传统搜索算法和普通改进算法.

关键词: 自适应遗传算法; 自适应交叉算子; 自适应变异算子; 测试用例生成

Automatic Testing-Case Generation Based on Adaptive Genetic Algorithm

LI Zhu

(Chongqing Jiaotong University, Chongqing 400074, China)

Abstract: In software testing, the key to a successful test is a fast and efficient testing-case generation. Genetic algorithm is an algorithm to search for the optimal solution by simulating the natural process of evolution. The algorithm guides the of search direction through the selection, crossover and mutation operations. to reach the global optimal solution step by step. Traditional genetic algorithm is widely used in the test case generation by many scientific researchers due to its better global search ability. But the genetic algorithm can easily lead to convergence to a local optimal solution because of its inherent defects “premature convergence”. In order to solve this problem, the author proposed an adaptive genetic algorithm in this paper. The crossover operator and mutation operator of the proposed algoritym can be adjusted automatically according to the change of the program. The improved algorithm is then applied in the test case generation process. The test results show that this algorithm is better than the traditional search algorithm and common improved algorithm in efficiency and effectiveness of testing-case generation.

Key words: adaptive genetic algorithm; adaptive crossover operator; adaptive mutation operator; testing-case generation

1 引言

随着计算机的普及和发展, 各类软件和程序迅猛发展, 人们对软件的需求和要求越来越高, 造成软件规模和复杂度大幅增加, 从而给软件测试带来了巨大困难. 因此, 如何有效快速的实现软件的测试就变得十分必要.

遗传算法是一种模拟自然界生物进化过程寻求最优解的一种算法. 因为遗传算法具有较好的全局搜索能力和较高的运转效率, 近年来, 一些研究人员尝试将遗传算法作为软件测试用例生成算法. 上海交通大学姚尧在其论文“一种基于遗传算法的软件测试用例生成新方法”中构造了一种遗传算法生成测试用例模型, 将遗传算法与软件测试用例生成有机结合起来, 提高了测试用例生成的效率; 王者思等 3 人提出了软件程序中的评价函数构造及插装方法, 进一步遗传算法应用于测试用例生成提供了依据. 然而, 遗传算法

① 收稿时间:2015-05-05;收到修改稿时间:2015-06-18

存在的“早熟收敛”[1]缺陷, 极易造成算法收敛于局部最优解, 而无法得到全局最优解(若是用于测试用例生成, 则为理想的测试用例).

本文对通过对遗传算法交叉算子和变异算子的改进, 使算法搜索方向可以根据适应度纸的变化进行自动调整, 可在一定程度上克服传统遗传算法全局搜索能力差、效率低下的问题. 将改进后算法应用到程序测试用例生成及优化中, 实验结果表明该算法在生成效率方面比传统搜索优化算法和非自适应的遗传算法改进方法更优.

2 遗传算法及其在软件测试中的应用

2.1 传统遗传算法

遗传算法是一种模拟生物进化过程[2], 通过指导种群进化方向得到最优种群的过程. 遗传算法中种群是由一系列染色体组成, 染色体是由一组数字表示, 该组数字中的每一位成为基因, 遗传算法就是通过基因的变化来指导种群的进化方向. 传统遗传算法描述[3]如下:

(1)根据问题需求, 将产生第一代种群.

(2)计算种群中个体的适应度值(适应度函数应根据问题需求产生).

(3)根据问题需求设置迭代次数和适应度阈值, 若迭代次数低于设定值或该种群中最高适应度值低于设定值, 进入循环操作: ①产生子代种群, 然后进行遗传算子(选择算子、交叉算子和变异算子)操作②计算新种群的适应度值.

(4)当新种群染色体适应度值或迭代次数达到设定值, 停止算法.

2.2 基于遗传算法生成测试用例的系统模型

为方便说明遗传算法如何生成测试用例, 文中给出该模型[4] (图 1). 从图中可以看出算法分两个部分: 算法执行部分和评价部分:

①执行部分: 初始化种群, 对染色体进行编码, 此时被测试的程序接收参数并运行评价部分; 算法执行部分根据评价部分参数适应度值的高低进行遗传算子操作, 产生子代种群, 循环往复, 直至覆盖路径的目标参数值被发现.

②评价部分: 在被测程序中插入评价函数, 实现对参数的评价.

该模型的重点就是将遗传算法与软件测试用例生成有机结合起来, 关键是在被测程序中插入被测评价函数, 形成对测试用例(被测程序)的评价体系, 利用评价函数(插桩程序)形成遗传算法适应度函数, 完成对被测程序的适应性评价. 该模型存在的主要问题是在算法运行后期, 由于遗传算法的早熟收敛的存在, 容易导致算法收敛于局部最优解, 无法得到全局最优解(理想的测试用例). 本文提出的自适应遗传算法就是在算法的后期, 通过对交叉算子和变异算子的自适应设计, 自动调整算法前进方向, 从而得到全局最优解.

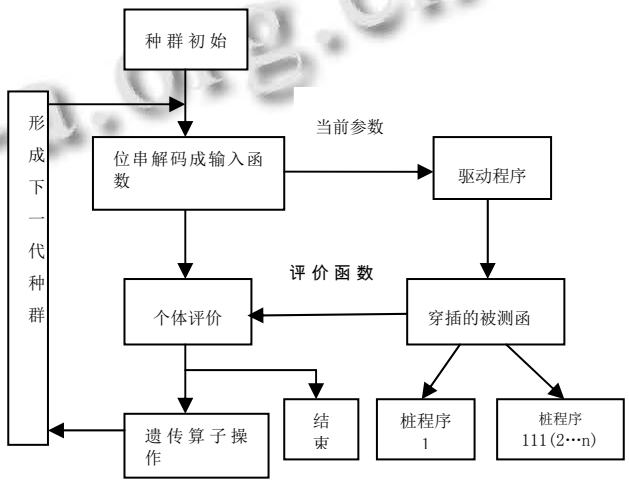

图 1 基于遗传算法测试用例生成的系统模型

3 传统遗传算法和非自适应遗传算法遗传算子设计

遗传算子是模拟自然界自然选择、杂交和基因突变等操作保证种群多样性而进行的, 包括: 选择、交叉和变异, 用以指导算法的前进方向. 传统遗传算法(又称基本遗传算法)和非自适应遗传算法遗传算子设计介绍如下.

3.1 传统遗传算法遗传算子设计

3.1.1 选择算子的设计

选择算子是根据自然界“优胜劣汰”理论总结得出的, 是遗传算法中起着重要的作用, 选择就是将种群中的优秀(适应度高)个体保留下来, 遗传到下一代; 适应度低的个体被保留下来的几率较低. 常用的个体选择方法有: 最优保存策略、排序选择、随机联赛选择和轮盘赌选择, 本文采用轮盘赌选择法.

轮盘赌选择法[5]是一种常用的随机选择法, 又称比例选择法, 个体被选中的概率与其适应度值(亦称优秀程度)的大小成正比. 假若 M 为种群大小, F_i 为第 i

个个体的适应度, 那么个体 i 被选中的概率表示为:

$$p_i=\frac{F_i}{\sum_{i=1}^{M}F_i},\quad \text{其中}i=1,2,....\text{M}$$

通过观察上面的式子可以发现, 个体适应度越低, 选中它的几率就越小, 反之, 就越大.

3.1.2 交叉算子的设计

基因的重新组合是自然界生物进化的核心流程, 因此, 交叉算子也就成为遗传算法的核心算子. 交叉就是对父代染色体中的部门基因互相替换重组, 然后产生两个不同的新染色体, 产生下一代. 这样极大的增加了染色体的多样性, 为遗传算法产生最优解, 提供了充分的样本. 传统遗传算法采用的交叉方式为单点交叉:

单点交叉[6](又称简单交叉), 是指在染色体位串中随机设置一交叉点, 将本染色体交叉点右侧的部分与另一染色体同一交叉点右侧部分进行交换, 左侧部分保持不变, 从而行成两个新的染色体位串.

3.1.3 变异算子的设计

在算法的后期, 种群的适应度高度接近, 通过交叉已不能有效改变种群的多样性, 此时, 变异算子就变的尤其重要. 所谓变异是指改变群体中个体串上的某些基因位上的基因值, 正式由于变异算子的存在, 算法后期才可以顺利地朝最优解靠近, 收敛加速. 通过交叉算子和变异算子的有效结合, 遗传算法在全局和局部空间都具有良好的搜索能力. 传统遗传算法采用的是基本位变异.

基本位变异算子[7]就是指随机指定个体基因串的某几个基因(大于等于 1 的整数)以变异概率 p_m 进行变换. 对于二进制编码个体, 若基因值为 1, 则变为 0; 若基因值为 0, 则变为 1.

3.2 非自适应遗传算法生成测试用例

3.2.1 选择算子设计

此处采用的选择算子与传统遗传算法相同.

3.2.2 交叉算子的设计

在遗传算法进化了一段时间后,群体往往会向搜索空间中包含较优解的区域收敛,这时群体中会存在较多相同的染色体,而传统的交叉算子在两个染色体相同时将失效,从而降低了算法的搜索能力,容易形成“早熟”. 为解决这一问题,我们在前面讨论的基础上给出自交叉算子这一新算子,其操作步骤如下:

(1)设染色体 X 被选中参与交叉,则生成该染色体的伴随染色体 Y,染色体 Y 的每一位都与 X 不同;

(2)染色体X和Y利用一般的一点交叉算子进行交叉运算,生成新染色体 X′和 Y′;

(3)任取新染色体 X′和 Y′中的一个取代原染色体 X.

3.2.3 变异算子的设计

此处, 非自适应遗传算法采用的变异方式为均匀变异[7]. 均匀变异是一种以较小概率, 用某范围内的随机产生的均匀分布的数来替换个体编码串中被选中的基因. 例如:

$M=m_1m_2,\ldots..m_{\text{k}}\ldots,m_n$, 其中 m_{k} 为变异点, 其所在范围为 $\left[Z^{k}_{\min},Z^{k}_{\max}\right]$, 那么在此变异点进行均匀变异后得到的新个体为: $M'=m_1m_2,\ldots..m_k^{'}\ldots,m_n$, $m_k^{'}=Z^{k}_{\min}+r\cdot(Z^{k}_{\max}-Z^{k}_{\min})$, 其中, r 为随机数, 满足[0, 1]范围内的均匀概率分布.

在整个搜索空间, 均匀变异使搜索点可以自由移动, 使得种群多样性增加, 快速的向算法最优解靠近.

4 自适应遗传算法遗传算子设计

本文所述自适应遗传算法, 就是通过对交叉算子和变异算子的设计, 使其根据程序变化自动调整两种算子的大小, 尽量使得算法跳出局部最优解找到全局最优解的一种算法.

4.1 适应度函数的构造

适应度函数的构造是遗传算法能够得到全局最优解的关键, 本文采用 TRACEY 提出的“分支函数插桩法”[8]得到适应度函数形式为: $F=\sum_{i=1}^{n}f_i$

其中, f_i 为插入在被测程序中的实值函数(实值函数插桩方法, 将在下文予以说明). F 为所有实值函数 f_i 的连加形式:

$F=F(f_1(x_1,x_2,...x_n),f_2(x_1,x_2,...x_n),...f_n(x_1,x_2,...x_n))$, 亦即 $F=\sum_{i=1}^{n}f_i$.

4.2 遗传算子的选择与改进

4.2.1 选择算子的设计

选择算子的设计, 本文采用 Whitley、Kuo 等专家提出的一种非单调适应值标度变换方法[9]. 设群体数量为 n 的群体 $\text{P}=\{\text{a}_1,\text{a}_2,\ldots,\text{a}_n\}$, 其中各个个体适

应值分别为 $f(a_1), f(a_2), \ldots f(a_n)$，$f_{ave}=\frac{\sum_{i=1}^{n} f_i}{n}$ 为平均适应度值. 该适应值的非单调标度变换设置为:

$f'(a_j)=|f(a_j)-f_{ave}|$, $a_j \in$P, j=1,2,…n

其中 $a_j \in$P, $f'(a_j)$ 值越大, 个体适应值偏离 f_{ave} 的染色体具更容易被选择.

4.2.2 自适应交叉算子和变异算子设计

为避免遗传算法“早熟收敛”缺陷, 本文提出自适应交叉算子(p_c)和自适应变异算子(p_m)概念, 采用适应度均值 f_{ave}、最大适应度 $f_{\max}$、最小适应度 $f_{\min}$ 三个参数表示种群适应度的集中与否. p_c 和 p_m 的调整方式如下:

$$p_c=\begin{cases} p_c\dfrac{1}{1-\dfrac{f_{\min}}{f_{\max}}}, & \dfrac{f_{ave}}{f_{\max}}>a, \dfrac{f_{\min}}{f_{\max}}>b, p_c<1-b, \\ p_c & \text{其他} \end{cases}$$

$$p_m=\begin{cases} p_m\dfrac{1}{1-\dfrac{f_{\min}}{f_{\max}}}, & \dfrac{f_{ave}}{f_{\max}}>a, \dfrac{f_{\min}}{f_{\max}}>b, p_m<1-b, \\ p_m & \text{其他} \end{cases}$$

①当 $f_{\min}$ 与 $f_{\max}$ 的值越接近说明遗传算法限于“早熟收敛”的可能性越高. 在此处, 我们设定一参数 b ($0<b<1$)当 $f_{\min}/f_{\max}>b$ 时, 视为种群较为集中. b 值越小, 表明集中程度越高.

②设定参数 a ($0.5<a<1$), 当 $f_{ave}/f_{\max}>a$ 时, 认为种群集中. a 接近 0.5, 判断为“集中”的可能性越高.

③当 $f_{ave}/f_{\max}>a$ 且 $f_{\min}/f_{\max}>b$ 时, 认为种群“集中”, 使 p_c 和 p_m 随着种群的集中程度的变化而自适应改变; 否则, 认为种群“分散”, 此时, 保持 p_c 和 p_m 初始阶段较小值.

④其中设定 $p_c<1-b$, $p_m<1-b$ 是为确保交叉和变异概率小于 1.

本文参数设定 a=0.78, b=0.24.

5 仿真实验

为验证该算法在测试用例生成方面的时效性, 本文通过下列程序段对被测程序分支函数和适应度函数构造、程序插桩过程[10]予以说明, :

```
void main(int input)     // L1(表明为程序的第一行)
{   int i;                          // L2
    for(i=0; i<N; i++)              // L3
    if(a[i]<0)                      // L4
     break;                         // L5
    if(i<N)                         // L6
    {   a[i]=input;                 // L7
      return True;                  // L8
    }                               // L9
   else                             // L10
    return false;                   // L11
  }                                 // L12
```

本文通过 Eclipse7 环境下采用 java 语言编程. 上述程序共 12 行, 分为 2 条路径: “L1-L3, L6, L 10, L 11, L 12”和“L1-L3, L4, L 5, L 6-L 9, L 12”. 其中路径“L1-L3, L6, L 10, L 11, L 12”包含三个分支语句: L4, L6, L10, 在该三条语句前插桩分支函数 f. 根据 Tracy“分支函数插桩”的要求, L3 语句运算符是“<”, 要保证该语句为真, 则分支函数 fi 的值须为零, f1=i-N; L4 为 f2=a[i]-0;L6 为 F3=i-N;因此, 适应度函数构造为三个分支函数相加的形式, 即: F=f1+f2+f3(fi 的值要根据分支判断的真假来选取). 上述程序段分支函数插桩及评价函数具体情况如下:

```
/* Line 1 */      void main(int input)
/* Line 2 */    { int i;
                    f1=i-N;          /*分支函数插桩*/
/* Line 3 */      for(i=0; i<N; i++)
                    f2=a[i]-0;     /*分支函数插桩*/
/* Line 4 */        if(a[i]<0)
/* Line 5 */        break;
                    f3=i-N;     /*分支函数插桩*/
/* Line 6 */       if(i<N)
/* Line 7 */       {   a[i]=input;
/* Line 8 */          return True;
/* Line 9 */       }
/* Line 10*/        else
/* Line 11*/       return false;
                   if(f1<0)   f1=0;
                     if(f2<0)   f2=0;
                     if(f3<0)   f3=0;
                     F=f1+f2+f3;     /*适应度函数*/
                     return F;
```

/* Line 12 */ }

表 1 中的数据是在程序段中插入桩程序, i 为循环次数, a[i]为设定的程序运行次数, 然后根据 i 和 a[i]的取值计算出桩程序的取值, 为求适应度函数值(评价函数)做准备. 在算法执行部分中只要 F 的值为零; 各变量取值就可满足测试路径 L1-L3, L4, L5, L6-L10, L12. 取值见下表(设定 N=6, i 和 a[i]的值任意):

表 1 插入桩程序数据表

i	a[i]	f1=i-N	f2=a[i]-0	f3=i-N
0	60	-6	60	-6
1	50	-5	50	-5
2	40	-4	40	-4
3	30	-3	30	-3
4	20	-2	20	-2
5	10	-1	10	-1

表 1 中 f1 和 f3 值为负, 因此由程序 f1 和 f3 的值应为零. 此时, 若 F=0, 满足测试路径, 系统可以很快生成该实例上的测试用例, 且拥有良好的收敛性. 在图 2 中将本文设计的基于自适应遗传算法的生成测试用例的方法与传统搜索优化算法生成测试用例的方法、非自适应遗传算法生成测试用例方法[11]在时间效率进行对比, 可以看出, 随着被测程序分支数增加, 本文提出的算法在时延上表现为线性增加, 而传统搜索优化算法则表现为指数增加的趋势; 相比非自适应遗传算法生成测试用例, 自适应遗传算法生成测试用例虽在初始阶段表现不够优秀, 但随着分支数的增加, 自适应遗传算法在时间效率上仍具有明显的优势, 收敛性更优秀[12].

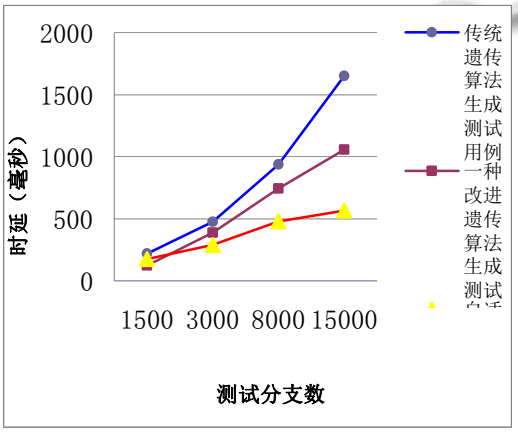

图 2 三种算法在生成测试用例时间效率上的对比

6 结语

本文对传统遗传算法交叉算子和变异算子方面进行了改进, 并将改进后的算法应用于一测试用例生成实例. 通过本文研究发现, 自适应交叉和变异算子可以较好地指导算法前进的方向, 并可在一定程度上避免算法限于“早熟收敛”, 提高了测试用例生成的效率, 效果明显优于传统搜索算法和非自适应改进型遗传算法. 当然我们也可以看到被测程序函数插桩是手工完成, 一旦程序复杂性过高, 会带来极大不便, 如何有效实现程序自动插桩是下一步急需解决的问题.

参考文献

1 付旭辉,康玲.遗传算法的早熟问题探究.华中科技大学学报(自然科学版),2003,31(1):53–55.
2 陈有青,徐蔡星,钟文亮,等.一种改进选择算子的遗传算法.计算机工程与应用,2008,44(2):44–45.
3 李敏强,寇纪淞,林丹等.遗传算法的基本原理及应用.北京:科学出版社,2002.
4 姚尧.一种基于遗传算法的软件测试用例生成新方法.计算机与数字工程,2009,37(1):18–21.
5 夏佳梅,曾建潮.一种基于轮盘赌选择遗传算法的随机微粒群算法.计算机工程与科学,2007,26(9):51–53.
6 Holland JH. Adaptation in Nature and Artificial Systems. MIT Press, 1992.
7 李延梅.一种改进的遗传算法及应用[硕士学位论文].广州:华南理工大学,2012.
8 Tracey N, Clark J, Mander K. The way forward for unifying test case generation: The optimization-based approach. International Workshop on Dependable Computing and its Applications. Johannesburg, South Africa. 1998. 169–180.
9 Kuo T, Hwang S. A genetic algorithm with disruptive selection, IEEE Trans. on Systems, Man and Cybernetics Cybernetics Part B: Cybernetics, 1996, 26(2): 65–69.
10 李柱,丁晓明.用于测试用例生成的遗传算法改进.科学技术与工程,2011,11(5):990–991.
11 荚伟,高仲仪.用遗传算法实现软件结构测试数据的自动生成.计算机与数字工程,1996.24(1):7–13.
12 梁艳春,王在申,周春光.选择和变异操作下遗传算法的收敛性研究.计算机研究与发展,1998,35(7):657–662.