ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.17, No.8, August 2006, pp.1688−1697
DOI: 10.1360/jos171688

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

一种基于 Messy GA 的结构测试数据自动生成方法*

薛云志[1,2+], 陈 伟[1,2], 王永吉[1,3], 赵 琛[1], 王 青[1]

[1](中国科学院 软件研究所 互联网软件技术实验室,北京 100080)

[2](中国科学院 研究生院,北京 100049)

[3](中国科学院 软件研究所 计算机科学重点实验室,北京 100080)

An Automated Approach for Structural Test Data Generation Based on Messy GA

XUE Yun-Zhi[1,2+], CHEN Wei[1,2], WANG Yong-Ji[1,3], ZHAO Chen[1], WANG Qing[1]

[1](Laboratory for Internet Software Technologies, Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China)

[2](Graduate School, The Chinese Academy of Sciences, Beijing 100049, China)

[3](Laboratory of Computer Science, Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China)

+ Corresponding author: Phn: +86-10-62565272, E-mail: yunzhi@intec.iscas.ac.cn, http://www.iscas.ac.cn

Xue YZ, Chen W, Wang YJ, Zhao C, Wang Q. An automated approach for structural test data generation based on Messy GA. *Journal of Software*, 2006,17(8):1688−1697. http://www.jos.org.cn/1000-9825/17/1688.htm

Abstract: Structural testing is one of the basic approaches for identifying test cases. Because of complexity of programming languages and variety of applications, an efficient approach to automated generation of structural test data is to breed search iteratively by profiling of program execution. Based on Messy GA, an automated approach for generating such data is proposed in this paper by solving $F(X)$, a test coverage function of test data set. It utilizes Messy GA's prominent feature that can optimize complicated problems without prior knowledge about schema arrangement in chromosomes, so that it can improve concurrency level of searching and test coverage. Compared with other approaches based on GA, the experimental results for several typical programs and real-world applications show that it can generate higher quality test data more efficiently, and can be applied to larger applications.

Key words: structural test; test data; test case; automatic generation; genetic algorithm; sizable chromosome; Messy GA

摘 要: 结构性测试是标识测试用例的基本方法之一.由于程序语言的复杂性以及被测程序的多样性,自动生成结构测试数据的一种有效方法是根据程序运行结果指导生成过程,通过不断迭代,生成符合要求的测试数据集.提出一种基于 Messy GA 的结构测试数据自动生成方法,将测试覆盖率表示为测试输入集 X 的函数 $F(X)$,并利用 Messy GA 不需要染色体模式排列的先验知识即可进行优化求解的性质对 $F(X)$进行迭代寻优,进一步提高了搜索的并行性,并最终提高测试覆盖率.对一组标准测试程序和若干实际应用程序的实验结果表明,较之现有

* Supported by the National Hi-Tech Research and Development Program of China under Grant No.2004AA1Z2100 (国家高技术研究发展计划(863)); the Program of the Chinese Academy of Sciences for One Hundred Talents under Grant No.BCK35873 (中国科学院百人计划)

Received 2005-07-07; Accepted 2005-12-01

基于遗传算法的生成方法,该方法能够以更高的效率生成更高质量的测试数据,并适用于较大规模的程序.
关键词: 结构测试;测试数据;测试用例;自动生成;遗传算法;变长度染色体;Messy GA
中图法分类号: TP311 文献标识码: A

结构性测试是软件测试活动中标识测试用例的基本方法之一[1−3],其自动生成方法可以分为随机测试数据生成、静态测试数据生成和动态测试数据生成[4].其中,随机测试数据生成方法不断随机生成测试数据,直到找到一个有用的输入或者达到规定的资源限制.该方法容易实现,针对简单的程序或测试策略可以取得较好的结果.其缺点是,对于较为复杂的程序或测试策略,几乎不可能以合理的成本生成足够的测试数据[4].但由于容易实现并且常在文献中引用,该方法可以为其他方法提供一个良好的比较基准.

静态测试数据生成方法对被测程序中的变量赋以符号值,将测试目标表达成约束条件下的算术表达式求解问题,利用其良好的数学性质来生成测试数据.求解方法主要有区间算术[5−7]、约束求解[8,9]等.在实际运用中,该方法遇到许多问题.文献[10]中指出,它“要求复杂的代数运算并且在处理数组和指针引用时存在困难”;文献[7]中也指出“在别名引用、不定长循环和符号表达式长度等方面存在实用上的困难”;文献[8]中进一步指出,在测试数据生成中,应用符号执行技术还面临着组合爆炸问题.

动态测试数据生成的基本思想是:反复执行被测程序,根据在被测程序运行过程中收集的信息来判断当前输入在多大程度上能够满足特定的测试需求,并以此指导生成过程[3,4,10−13].借助这种反馈机制,测试输入能够逐步被修正直至满足测试需求,而不必借助于程序内部结构和行为的数学表示,从而避免静态测试数据生成的问题[4].基本做法是,将测试目标按照测试策略要求表达成一个函数,该函数在该测试目标被满足时取得最小值.这样,测试数据生成问题被转化为函数最小化问题,其重点在于选择合适的目标函数和函数最小化方法.比较有代表性的最小化方法是梯度下降法[3],但梯度下降法不是全局优化方法,可能会在遇到局部极小点时失效[13].因此,比较理想的策略是使用全局优化方法来完成函数最小化.

遗传算法是一种基于演化的概率性全局优化算法[14],其主要特点是对优化对象没有可导和连续性的要求,具有隐并行性和全局寻优能力,可以自动获取搜索过程中的有关信息并用于指导优化[15].因此,许多研究工作采用遗传算法来完成函数最小化,文献[4,13]结合覆盖表将目标路径表达成一个最小化函数,采用微分式遗传算法进行全局寻优;文献[16]将输入空间划分成多个范围,根据待选输入的多种特性组成一个最小化函数,使用简单遗传算法进行求解,并使用求解过程中的化石记录来指导求解过程;文献[17]将遗传算法和形式化概念分析(formal concept analysis)相结合,以提供对测试数据与相应的测试执行间关系的追踪;文献[18]列举了近年来在基于遗传算法的测试数据生成方法中的进展,包括使用系统判据、模型、仿真系统等替代真实系统,以避免反复执行真实系统带来的开销,以及在各个遗传算子上所作的改进.

上述工作利用了遗传算法的并行性和全局寻优能力,在搜索过程中引入并行性,提高了搜索效率和准确性.文献[2]进一步提出了一种面向覆盖率目标的测试方法,将测试输入集的覆盖率作为测试输入集的函数,记为$F(X)$,通过对此函数寻优来指导遗传算法的迭代过程.由于 $F(X)$同时描述了测试策略所要求的多个测试目标,因此对$F(X)$的寻优相当于寻找一个测试输入集,以同时满足多个测试目标.一般来说,$F(X)$没有解析形式.对没有解析形式的函数进行优化是很困难的,只有当了解被测程序在所有可能输入下的相应输出后(这恰好是结构测试数据生成所要解决的问题),才能得到 $F(X)$的解析形式.因此,对函数 $F(X)$的寻优,需要寻找等价于 $F(X)$的其他具有解析形式的函数,或者选择不依赖函数形式的优化方法.

文献[2]通过将 $F(X)$转化为对控制依赖图中边的覆盖率来获得等价的具有解析形式的函数,本文则利用 Messy GA 不需要染色体模式排列的先验知识即可求解复杂问题的性质对 $F(X)$进行优化.不同于简单遗传算法中必须为每一个基因座指定值,Messy GA 只要求在染色体中的若干基因座有相应的值就可进行求解.这一性质使得我们可以利用每次迭代过程中生成的测试数据对 $F(X)$进行寻优,而不必等到所有测试数据都生成之后.另一方面,对测试输入集 X 的编码使得迭代过程中产生的优良个体均得以保留,在下一轮迭代中可以同时搜索这些优良个体附近的解空间,提高了搜索的并行性.对一组测试程序和若干实际应用程序的实验结果表明,较之现

有的基于遗传算法的生成方法,该方法能够以更高的效率生成更高质量的测试数据,并适用于较大规模的程序.

本文第 1 节介绍 Messy GA 基本原理.第 2 节讨论使用 Messy GA 进行结构测试数据生成时的具体问题,包括算法步骤、编码方式、适应度函数的设计、算子设计以及相应的算法步骤等.第 3 节通过实验讨论算法性能,及算法参数的不同对性能造成的影响.第 4 节是结论及后续研究工作的设想.

1 Messy GA 算法原理

Messy GA 是一种典型的变长度染色体遗传算法[16],通过不断组合相对较短的、具有较高适应度的染色体片断来形成更长、更复杂的染色体,以找到问题的最优解[19].与使用固定长度染色体的简单遗传算法相比,Messy GA 可以不借助优良个体中的染色体中模式排列的先验知识来进行优化,并在染色体一旦增长到足够长度之后,只需很少的迭代次数就可以发现最优解,并且寻优过程可以"提高优良的染色体片断在群体中的比例,加速优良染色体片断的传播速度"[20],从而获得比简单遗传算法更好的性能.

1.1 编码、目标函数和算子

将简单遗传算法的染色体编码传中各基因座位置及相应基因值组成一个二元组,并按照一定顺序排列起来,就组成了 Messy GA 的染色体编码.一条染色体一般可以表示为

$$\{(i_1,v_1)\ (i_2,v_2)\ldots(i_k,v_k)\ldots(i_n,v_n)\},$$

其中,n 是一条染色体中值的数量,i_k 是所描述基因编号,v_k 是其对应的基因值.

在这种编码方式中,i_k 可能会重复,也不一定连续.通过这种方式,可以同时描述不同的模式.例如,某一条染色体为{(1,0) (2,0) (4,0) (4,1)},则其表示在位置 1 有一个值 0,在位置 2 有一个值 0,在位置 3 没有值,在位置 4 有两个分别为 0 和 1 的值,对应的模式分别是 00*0 和 00*1.

按照上述方式编码染色体后,可能为同一个基因座指定多个值,也有可能不指定任何值.因此,需要使用不同于简单遗传算法的策略来计算染色体的适应度函数值.文献[19]中对于为一个基因座指定了多个值的情形,只考虑出现的第 1 个值;对于没有为某个基因座指定值的情形,使用一种称为竞争性模板的方法来填充缺失的信息:用使用局部寻优算法(爬山法等)获得的局部最优解中相对应的信息来填充.正是这种不为每个基因座指定唯一值的做法,使得 Messy GA 可以不依赖于染色体中模式排列的先验知识.

由于编码方式的特殊性,简单遗传算法中的交叉(杂交)算子不适用于 Messy GA,而用切断算子(cut operator)和拼接算子(splice operator)代替[19]:

- 切断算子:以特定概率随机选择一个位置,在该处将个体切断,使其成为两个个体的基因型;
- 拼接算子:以特定概率将两个个体的基因型拼接在一起,使其合并成为一个新个体的基因型.

1.2 算法步骤

Messy GA 的基本步骤可以简单描述如下:

(1) 初始化.设定变异率为μ,群体尺寸为 M,初始染色体长度为 k,演化代数为 T,产生出所有长度为 k 的子串作为初始群体 $P(0)$.

(2) 基本处理阶段(primordial phase).对群体 $P(0)$使用选择算子,以保留差异较大的个体.

(3) 并列处理阶段(juxtapositional phase)

a) 对群体 $P(t)$使用变异算子、切断算子和拼接算子,以生成新的个体;

b) 评价每个个体的适应值;

c) 通过选择算子使群体尺寸保持不变.

d) 重复上述步骤,直到满足终止条件为止.

Messy GA 的处理过程分为两个阶段:基本处理阶段和并列处理阶段.在基本处理阶段,通过选择算子,使得群体中只包含差异度较大的个体(即个体之间具备的不同基因达到一定数量).在并列处理阶段,通过选择算子、变异算子(文献[19]中没有使用该算子)、切断算子和拼接算子来产生新的个体.在计算出相应的适应度函数值

以后,将这些新个体加入到群体当中,并通过选择算子使得群体中只包含具有较高适应度的个体.整个处理过程是一个迭代过程,算法中的并列处理阶段不断迭代,直到满足终止条件为止.

2 基于 Messy GA 的结构测试数据生成方法

设测试输入集为 X,被测程序为 S,对应的输出集为 $S(X)$,选定的测试覆盖策略为 δ,对应的测试覆盖率为 Σ,则 Σ可以表达为

$$\Sigma=F(X,S,S(X),\delta).$$

显然,在给定被测程序 S、测试覆盖策略 δ的前提下,上述函数的解析形式只有在获得每个测试输入所对应的输出之后才能得到.穷举测试输入是不可能的,而测试输入的选择恰好是结构测试数据生成所需要解决的问题;另一方面,很难对没有解析形式的函数进行优化.Messy GA 由于其染色体中特殊的模式排列方式,不需要获得模式排列的先验知识就可以进行优化[20],这一性质使得不依赖于 $F(X,S,S(X),\delta)$的解析形式就可以对此函数进行优化.

本文使用一个被相关文献所广泛采用的三角形分类程序 TriType 作为示例.该程序接受 3 个分别代表三角形某一条边的长度的整数,判断这 3 个整数是否能够组成一个合法的三角形,并进而判断出三角形的类型.示例程序用 Java 实现,共包含了 15 个条件、17 个判定.

2.1 编码方式

在基于 Messy GA 的测试数据生成方法中,一条染色体包含了一个测试输入集中一个参数的全部取值情况.若干条这样的染色体(其数目取决于输入参数的个数)组成一个个体,而若干个个体组成一个种群.染色体的编码采用实数编码,位置值用基因在染色体中的对应位置隐含表示(因此,测试数据必须用序列方式而不是集合方式来表示).在这种编码方式下,个体中各个染色体上的等位基因组成一个测试输入.

一条染色体 C 可以表示为

$$C=\langle V_1,V_2,\ldots,V_k\rangle,$$

其中:k 为染色体的长度;V_i 为一整数(或其他类型,取决于输入参数的类型),表示某一基因的实际取值.一个个体 I 可以表示为

$$I=\langle C_1,C_2,\ldots,C_n\rangle=\langle\langle V_{11},V_{12},\ldots,V_{1k}\rangle,\langle V_{21},V_{22},\ldots,V_{2k}\rangle,\ldots,\langle V_{n1},V_{n2},\ldots,V_{nk}\rangle\rangle,$$

其中,n 为输入参数的个数,序列$\langle V_{1j},V_{2j},\ldots V_{nj}\rangle$表示第 j 个测试输入.

图 1 是对 TriType 的编码示例.其中:$\langle\langle 23,78,34,5,87,\ldots\rangle,\langle 2,-46,456,345,40,\ldots\rangle,\langle -34,76,320,83,94,\ldots\rangle\rangle$ 表示一个独立个体,即一个测试数据集;序列$\langle 23,78,34,5,87,\ldots\rangle$表示一个测试数据集中参数 a 的所有可能取值;序列$\langle 34,456,320\rangle$则表示了一个测试输入.

这种编码方式有两个优点:1) 在生成数据过程中的某一个时刻,之前所产生的所有可以较好地覆盖被测代码的测试数据(个体的染色体)都被保存在群体当中.因此,下一轮的迭代搜索可以同时搜索解空间中的多处位置,较之文献[4]中编码中仅包含一个测试输入的方式,该编码方式进一步提高了搜索的并行性;2) 只需要知道测试输入中的参数个数和相应类型即可确定编码,适用面较宽.

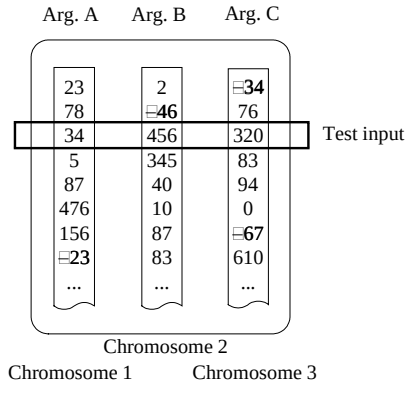

Fig.1 Individuals expression based on Messy GA

图 1 基于 Messy GA 的个体表示

2.2 覆盖表与适应度函数

为了度量被测程序的覆盖率,本文采用了 QUEST/Ada 系统中的覆盖表[21],见表 1.覆盖表中的每一条记录都

有一个类型域,表明该记录的类型(语句、分支、条件、判定、循环、路径).每个记录分别具有一个 TRUE 域和一个 FALSE 域,分别表示该记录是否被当前测试输入集所覆盖.在一次测试执行中一般不会出现所有的记录类型,而只是出现与测试目标相关的记录类型.例如,若测试目标是分支覆盖,则有效记录类型是分支类型;若测试需求是分支-判定覆盖,则有效记录类型是分支和判定类型.表 1 是当测试目标为条件-判定覆盖时一个可能的覆盖表,其中✓表示该域被覆盖,×表示该域未被覆盖,–则表示该域不适用.

Table 1 A possible coverage table as test criteria is C-D coverage

表 1 当测试需求是条件-判定覆盖时,一个可能的覆盖表

No.	Type	TRUE	FALSE
1	Condition	✓	×
2	Condition	×	✓
3	Decision	✓	–
4	Condition	×	✓
…	…	…	…

覆盖表中的记录可以用来计算个体的适应值:覆盖表中被覆盖的区域个数与所有区域个数之比即为个体的适应值.以测试目标为条件-判定覆盖为例,被测程序中的条件个数为 c,判定个数为 d,被覆盖的判定个数为 m, TRUE 域被覆盖的条件个数为 a, FALSE 域被覆盖的条件个数为 b,则适应度值 f 为

$$f=\frac{a+b+m}{2c+d}.$$

覆盖表以及覆盖结果可以通过对被测试代码打桩及运行打桩后的代码来获得.覆盖表当用户仅关心被测程序特定特征(例如某个条件或某条路径是否被覆盖)时不够有效,但当用户关心在某种测试覆盖指标下的覆盖率时,覆盖表可以在很大程度上提高搜索效率[4,21].采用覆盖表可以避免对源程序进行复杂的符号分析,从而可以适用于较大规模的程序[4].

2.3 切断算子和拼接算子

切断算子和拼接算子可将群体中各个个体重组,即将不同测试输入集中对应于同一输入参数的染色体进行重新组合,以利用较短的染色体(较少的测试数据)来达到较高的覆盖率.如图 2 所示.

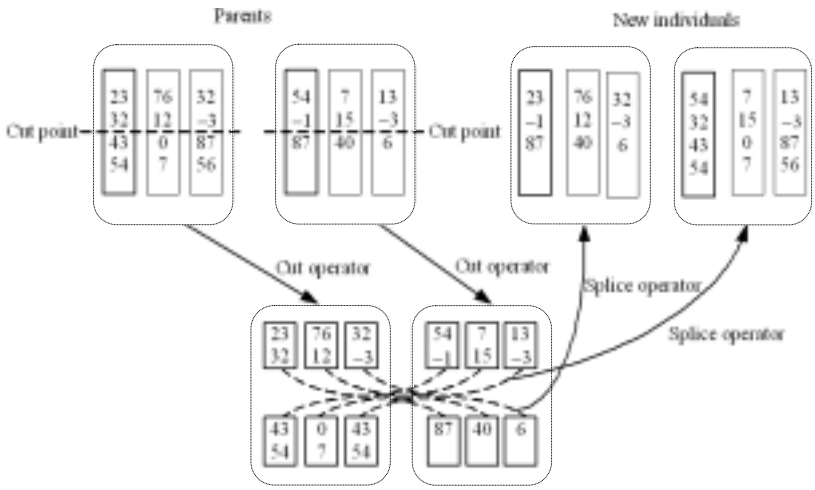

Fig.2 Employ cut operator & splice operator to produce new individual for TriType

图 2 在 TriType 中,使用切断算子和拼接算子产生新个体

首先,切断算子将群体中所有个体按照特定概率切分成两个部分:以特定概率在染色体中选择一个位置,在该处将染色体的基因型切断,使其成为两个新个体的染色体片断.设一个个体为

$$I_i^{\alpha}=\langle C_{i1},C_{i2},\ldots,C_{in}\rangle=\langle\langle V_{i11},V_{i12},\ldots,V_{i1k}\rangle,\langle V_{i21},V_{i22},\ldots,V_{i2k}\rangle,\ldots,\langle V_{in1},V_{in2},\ldots,V_{ink}\rangle\rangle.$$

切断算子确定一个位置 p,$1\le p\le k$,在 p 处将 I_i 切断为α部分和β部分:

$$I_i^{\alpha}:\langle\langle V_{i11},V_{i12},\ldots,V_{i1p}\rangle,\langle V_{i21},V_{i22},\ldots,V_{i2p}\rangle,\ldots,\langle V_{in1},V_{in2},\ldots,V_{inp}\rangle\rangle,$$

$$I_i^{\beta}:\langle\langle V_{i1(p+1)},V_{i1(p+2)},\ldots,V_{i1k}\rangle,\langle V_{i2(p+1)},V_{i2(p+2)},\ldots,V_{i2k}\rangle,\ldots,\langle V_{in(p+1)},V_{in(p+2)},\ldots,V_{ink}\rangle\rangle.$$

记这种关系为 $I_i=\langle p, I_i^{\alpha}, I_i^{\beta}\rangle$.

拼接算子则以特定概率随机选取两个被切断算子作用后的个体,交换各自的α和β部分,从而形成新个体.如拼接算子选定的两个个体为 I_i 和 I_j,长度分别为 k 和 m,分别被切割为$\langle p, I_i^{\alpha}, I_i^{\beta}\rangle$和$\langle q, I_j^{\alpha}, I_j^{\beta}\rangle$,交换各自的$\alpha$部分和$\beta$部分后为

$$I_i'=\langle p, I_i^{\alpha}, I_j^{\beta}\rangle,\quad I_j'=\langle q, I_j^{\alpha}, I_i^{\beta}\rangle.$$

即

$$I_i':\langle\langle V_{i11},V_{i12},\ldots,V_{i1p},V_{j1(q+1)},\ldots,V_{j1m}\rangle,\langle V_{i21},V_{i22},\ldots,V_{i2p},V_{j2(q+1)},\ldots,V_{j2m}\rangle,\ldots,\langle V_{in1},V_{in2},\ldots,V_{inp},V_{jn(q+1)},\ldots,V_{jnm}\rangle\rangle,$$

$$I_j':\langle\langle V_{j11},V_{j12},\ldots,V_{j1q},V_{i1(p+1)},\ldots,V_{i1k}\rangle,\langle V_{j21},V_{j22},\ldots,V_{j2q},V_{i2(p+1)},\ldots,V_{i2k}\rangle,\ldots,\langle V_{jn1},V_{jn2},\ldots,V_{jnq},V_{in(p+1)},\ldots,V_{ink}\rangle\rangle.$$

由于染色体的长度是有意义的(表示当前测试输入集中对特定参数的全部输入的个数),基因位置也是有意义的(各染色体的等位基因组成一个测试输入),因此对同一个个体,切断算子必须将所有染色体都在同一个位置切断,否则拼接算子拼接时可能因为片断长度不一而形成无效个体.同样,对应于不同参数的染色体之间的拼接是没有意义的,参与拼接的染色体片断必须对应于同一个参数.

2.4 变异算子

本文所提出的方法存在两个变异算子:第 1 个变异算子用来搜索当前测试输入的邻近区域,记为 A-op1;第 2 个变异算子用来在一个个体的所有染色体尾部添加新的基因,也就是增加新的测试输入,记为 A-op2.

A-op1 类似于简单遗传算法中的变异算子,在染色体复制过程中随机改变某一位基因的取值,以搜索候选解在解空间中的邻近区域.该操作随机选取一些染色体,并以特定概率确定一个位置,对选中的染色体的基因值使用几何漂移方法进行变异.如某条被选择进行变异的染色体为

$$C=\langle V_1,V_2,\ldots,V_k\rangle,$$

确定在第 p 个位置进行变异操作,漂移度为 0.989,则变异后的染色体为

$$C'=\langle V_1,V_2,\ldots,V_p\times 0.989,\ldots,V_k\rangle.$$

A-op2 在染色体的尾部添加新基因,从而增加染色体的长度(增加新的测试输入),以期获得更高的覆盖率.与 A-op1 不同,考虑到测试输入的合法性,A-op2 对个体内的所有染色体同时在尾部添加一个新基因.如一个个体为

$$I=\langle\langle V_{11},V_{12},\ldots,V_{1k}\rangle,\langle V_{21},V_{22},\ldots,V_{2k}\rangle,\ldots,\langle V_{n1},V_{n2},\ldots,V_{nk}\rangle\rangle.$$

若 A-op2 算子确定对此个体作用,则形成的新个体为

$$I'=\langle\langle V_{11},V_{12},\ldots,V_{1k},V_{1(k+1)}\rangle,\langle V_{21},V_{22},\ldots,V_{2k},V_{2(k+1)}\rangle,\ldots,\langle V_{n1},V_{n2},\ldots,V_{nk},V_{n(k+1)}\rangle\rangle.$$

由于新添加的基因(测试输入)是为了覆盖尚未被覆盖的区域,因此,可以使用与相应区域相关的启发式规则来初始化:

- 如果判定中出现等于操作符,则使用相等的随机数来初始化;
- 如果判定中出现大于(等于)操作符,则使用从大到小排列的随机数序列来初始化;
- 如果判定中出现小于(等于)操作符,则使用从小到大排列的随机数序列来初始化.

这样初始化后,新基因位于可使未覆盖区域覆盖的可行解的附近,只需经过很少的演化就可以找到可行解.

2.5 生成算法

基于 Messy GA 的结构测试数据生成方法是一个基于群体的迭代过程,利用 Messy GA 重组染色体中的优良片断以寻找最优解的性质,组合现有测试数据并利用变异操作引进新数据,以获得对被测程序相应的覆盖表较高的覆盖率.其步骤如下:

(1) 分析被测程序,并作可测试性转换[22];

(2) 根据测试覆盖策略、被测程序和转换后的被测程序建立覆盖表,确定适应度评价函数;
(3) 对转换后的被测程序打桩,在每一个分支处插入输出语句;
(4) 设定算法参数,包括群体大小、变异率等;
(5) 初始化群体,设定当前群体为 *P*;
(6) 使用群体 *P* 中的每个个体中的数据执行打桩后的程序,并收集在步骤(3)中插入的语句的输出信息;
(7) 根据个体运行之后的相应覆盖表的结果,评估个体的适应度:若满足终止条件,则算法结束;否则,执行下一步;
(8) 选择适应度较高的个体,组成新群体的一部分,称为 SUB;
(9) 对 SUB 中的个体进行切断和拼接操作,产生一部分个体;
(10) 对 SUB 中的个体进行 A-op1 操作,产生一部分个体;
(11) 对新群体中的所有个体进行 A-op2 操作,使得有一部分个体的长度有可能增加;
(12) 设新群体为 *P*,执行步骤(6).

其中,拼接算子和变异算子使得不必为每一条染色体预先指定所有的基因排列模式,而是可以在迭代过程中根据覆盖结果和被测试程序特征逐步填充.处理过程不再分为基本处理阶段(列举所有可能的候选解)和并列处理阶段(迭代演化出最优解),而是从初始群体出发,逐步迭代搜索最优解.其原因是,结构测试数据生成的搜索空间通常都非常大,列举出候选解集合所需耗费的资源是不可接受的.

3 实验结果讨论

3.1 覆盖率数据

本文采用了在相关文献中广泛使用的一组测试程序进行实验,包括:二分搜索、中位数计算、日期计算、插入排序、欧几里德最大公约数、三角形分类.这些程序具备相当的代表性,实际应用中的多数程序具有相似的复杂度.另外,从一个 Web 应用中随机抽取了 3 段代码作为测试程序,这些程序都具有较前述测试程序更高的复杂度,每个程序分支和判定数目之和均在 100 左右.设定算法参数为:群体尺寸为 100,并且允许算法在适应度函数取值没有明显变化的情形下最多演化 10 代,最多允许演化 1000 代,变异率为 0.001.测试策略为条件-判定覆盖,取 5 次运行结果的算术平均值作为其覆盖率.为了作出客观比较,参数来自文献[4].各种算法的对比数据见表 2(黑体部分数据来自文献[4]).

Table 2 Comparison of C-D coverage through data generated by various methods
表 **2** 各算法生成数据的 C-D 覆盖率对比

Algorithm	Random (%)	Simple GA (%)	GADGET[4] (%)	Messy GA (%)
Binary search	**80**	**70**	**100**	100
Computing the median	**100**	**100**	**100**	100
Number of days between two dates	**87.5**	**100**	**100**	100
Insertion sort	**100**	**92.9**	**100**	100
Euclidean greatest common denominator	**100**	**100**	**100**	100
Angle classification	**43.4**	**86.7**	**93.3**	100
Sample program 1	44.3	72.6	90.6	98.1
Sample program 2	40.1	66.2	89.2	94.1
Sample program 3	54.1	72.9	96.5	100

上述实验数据表明,在同等参数条件下,就测试覆盖指标为 C-D 覆盖而言,本文提出的基于 Messy GA 的方法与随机测试数据生成算法和基于简单遗传算法的测试数据生成方法相比,具有非常明显的优势.与文献[4]中所提出的 GADGET 方法相比,也能获得更高质量的数据.

3.2 参数对算法性能的影响

群体尺寸的大小和变异率会对遗传算法的性能和准确性造成明显的影响,本文设计一系列实验,试图确定这两个因素对基于 Messy GA 的测试数据生成方法的影响.所采用的样本程序为上述的 TriType 程序,设测试覆

盖指标为条件-判定覆盖.

设定变异率为 0.001,设定迭代 10 代后仍无明显改进则终止算法,最大演化代数为 1 000.群体尺寸为 10~200,步长为 10,对每个尺寸分别运行 5 次,取 5 次运行的算术平均值作为结果.图 3(a)为条件-判定覆盖率和最优解出现时间随群体尺寸大小的变化曲线.图 3(a)中数据表明:当群体尺寸较小(≤80)时,算法难以达到 100%的覆盖率;当群体尺寸过大(≥140)时,算法不稳定;只有当群体尺寸处在合适的范围([80,140])内时,算法才能稳定地获得 100%的覆盖率.另外,在算法表现稳定的阶段(群体尺寸在[10,130]内),群体尺寸的增大都有助于更快地获得最优解.例如,当群体尺寸为(40,50,60,70,80)其中之一时,算法所取得的覆盖率都为 93.3%,但最优解出现的时间分别为(554,468,424,326,261).这表明在一定范围内,群体尺寸的增大不仅可以提高获得最优解的机会,也可以显著提高获得最优解的速度,但过大的群体尺寸会使算法不够稳定.

设定群体尺寸为 100,设定迭代 10 代后仍无明显改进则终止算法.设定变异率为 0.0001~0.002,步长为 0.0001,对每个变异率分别运行 5 次,取其算术平均值作为结果.图 3(b)为条件-判定覆盖率和最优解出现时间随变异率的变化曲线.图 3(b)中数据表明,当变异率较小(≤0.0009)时,算法不能获得最高覆盖率;当变异率较大(≥0.0014)时,算法获得最高覆盖率的能力急剧下降;当变异率处于合适的区间([0.0009,0.0013])内时,算法能够稳定地获得最高覆盖率.图中曲线也表明:在一定范围内([0.0001,0.0013]),变异率的提高有助于加快获得最优解的速度.因此在一定范围内,变异率的提高不仅可以提高获得最优解的机会,也可以显著提高获得最优解的速度;但过高的变异率会降低算法获得较高覆盖率的能力.

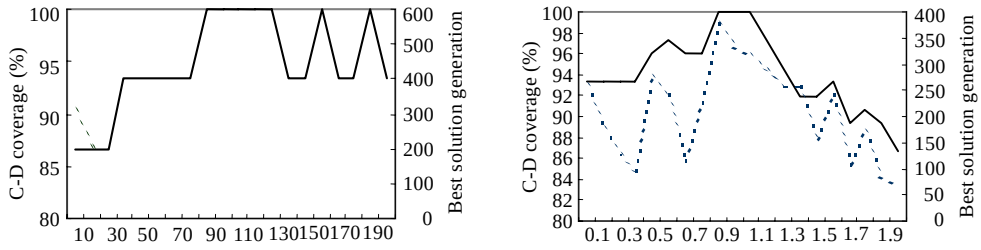

Fig.3 The influence of population size and aberrance ratio on performance of the algorithm

图 3 群体尺寸和变异率对算法性能的影响

3.3 算法性能讨论

本文以 TriType 为测试对象进一步比较了几种算法求得各自的最优解所需要的演化代数,以比较各种算法的时间效率.对 3 种以遗传算法为基础的方法,设定算法参数同第 3.1 节,均来自文献[4].并且对各种方法使用相同的初始群体.比较结果如图 4 所示.

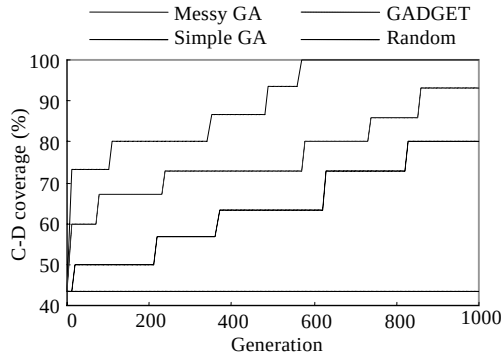

Fig.4 For TriType, comparison of generation needed by Messy GA, GADGET, simple GA and Random

图 4 对 TriType,Messy GA,GADGET,简单遗传算法和随机生成算法求解代数比较

图 4 中,横轴坐标为演化代数,纵轴坐标为 C-D 覆盖率.其中:随机数据生成方法作为比较的基准,其可获得的覆盖率不随演化代数的变化而变化,但其覆盖率最低(43.4%);简单遗传算法的覆盖率有所提高(86.7%),但其需要的演化代数却急剧增大,在 5 次运行中,有 4 次甚至在 1000 代的演化次数以内都不能得到其最优解.GADGET 方法可以获得较好的覆盖率(93.3%),但需要较多的演化代数.相比之下,本文所提出的基于 Messy GA 的方法由于可以同时搜索解空间中的多个可能区域,可以在更短时间内获得更高的测试覆盖率.

4 结论及未来工作

本文提出了一种基于 Messy GA 的结构测试数据自动生成方法,该方法将测试覆盖率表示为测试输入集的函数,利用 Messy GA,不借助染色体中模式排列的先验知识就可以对复杂问题进行优化的性质,对该函数寻优以获得优化的测试数据集.该方法将整个测试输入看作一个个体,其一条染色体包含了该输入集中一个参数的全部取值情况,通过对整个测试输入的评估来指导生成过程.

该方法对测试输入集 X 的编码使得迭代过程中产生的优良个体均得以保留,并可以同时搜索解空间中的多个可行解.与仅包含一个可行解的编码方式相比,进一步提高了测试数据生成过程中的并行性,从而加快搜索过程,以较低的成本生成满足测试覆盖目标要求的测试数据.另外,覆盖表的采用使得不要求了解代码中变量取值情况以及变量依赖关系,因此可以适用于较大规模的被测程序.对一组测试程序和若干实际应用程序的实验结果表明,较之现有的基于遗传算法的生成方法,该方法能够以更高的效率生成更高质量的测试数据,并对实际应用中较大规模的程序取得良好的效果.

未来工作主要有以下几个方面:

1) 初始群体的生成.该方法目前的初始群体是随机初始化的.若能利用从已有的某种方法获得的较好的测试数据组成初始群体,则数据生成速度应该可以提高.

2) 染色体长度的增减.该方法目前的染色体长度只是单调增加的,并不会减少.可以考虑在迭代过程中引入对单个测试输入的评估,若某个测试输入可被其他更好的测试输入所蕴含或覆盖区域小于某个阈值,则从染色体中去掉相应的等位基因.这样生成的染色体长度可能更短,相应的测试数据集也就更小.

3) 算法参数的自适应选取.实验数据表明:当参数处于某一个区间时,算法可以取得较好的性能,但是否可以根据被测试程序的特征选取合适的参数以获得更好的性能则值得研究.

References:

[1] Jorgenson PC. Software Testing: A Craftsman's Approach. 2nd ed. Beijing: Machine Press, 2003. 5−7 (in Chinese).

[2] Pargas RP, Harrold MJ. Test data generation using genetic algorithms. Journal of Software Testing, Verification and Reliability, 1999,9(4):263−282.

[3] Korel B. Automated software test data generation. IEEE Trans. on Software Engineering, 1990,16(8):870−879.

[4] Michael CC, McGraw GE, Schatz MA. Generating software test data by evolution. IEEE Trans. on Software Engineering, 2001,27(12):1085−1110.

[5] Sy NT, Deville Y. Consistency techniques for interprocedural test data generation. ACM SIGSOFT Software Engineering Notes, 2003,28(5):108−117.

[6] Chen T, Tse T, Zhouz Z. Semi-Proving: An integrated method based on global symbolic evaluation and metamorphic testing. ACM SIGSOFT Software Engineering Notes, 2002,27(4):191−195.

[7] 10Offutt A, Jin Z, Pan J. The dynamic domain reduction procedure for test data generation. Software Practice and Experience, 1999, 29(2):167−193.

[8] Zhang J, Xu C, Wang X. Path-Oriented test data generation using symbolic execution and constraint solving techniques. In: He JF, Yang FQ, eds. Proc. of the Int'l Conf. on Software Engineering and Formal Methods. Beijing: IEEE Computer Society Press, 2004. 242−250.

[9] DeMillo RA, Offutt AJ. Constraint-Based automatic test data generation. IEEE Trans. on Software Engineering, 1991,17(12): 900−910.

[10] Gupta N, Mathur AP, Soffa ML. Automated test data generation using an iterative relaxation method. ACM SIGSOFT Software Engineering Notes,1998,23(6):231−244.

[11] Korel B. Automated test data generation for programs with procedures. ACM SIGSOFT Software Engineering Notes, 1996,21(3):209−215.

[12] Gallagher MJ. Narasimhau VL. ADTEST: A test data generation suite for Ada software systems. IEEE Trans. on Software Engineering, 1997,23(8):473−484.

[13] Michael C, McGraw G, Schatz MA. Opportunism and diversity in automated software test data generation. Technical Report RSTR-003-97-13, RST Corporation, 1997. 1−17.

[14] Melanie M. An Introduction to Genetic Algorithms. Boston: MIT Press, 1998. 121−124.

[15] Shi ZZ. Knowledge Discovery. Beijing: Tsinghua University Press, 2002. 266−286 (in Chinese).

[16] Berndt D, Fisher J, Joshson L. Breeding software test cases with genetic algorithms. In: Sprague RH, ed. Proc. of the Int'l Conf. on System Sciences. Big Island: IEEE Computer Society Press, 2003. 338a.

[17] Khor S, Grogono P. Using a genetic algorithm and formal concept analysis to generate branch coverage test data automatically. In: Grünbacher P, Wiels V, Stirewalt K, eds. Proc. of the Int'l Conf. on Automated Software Engineering. Linz: IEEE Computer Society Press, 2004. 346−349.

[18] Berndt DJ, Watkins A. Investigating the performance of genetic algorithm-based software test case generation. In: Ramamoorthy CV, ed. Proc. of the Int'l Symp. on High Assurance Systems Engineering. Tampa Florida: IEEE Computer Society Press, 2004. 261−262.

[19] Kargupta H. The gene expression messy genetic algorithm. In: Proc. of the Int'l Conf. on Evolutionary Computation. Nagoya: IEEE Computer Society Press, 1996. 814−819.

[20] Zaritsky A, Sipper M. The preservation of favoured building blocks in the struggle for fitness: The puzzle algorithm. IEEE Trans. on Evolutionary Computation, 2004,8(5):443−455.

[21] Deason WH, Brown DB, Chang KH, Cross II JH. A rule-based software test data generator. IEEE Trans. on Knowledge and Data Engineering, 1991,3(1):108−117.

[22] Harman M, Hu L, Hieros R, Wegener J, Sthamer H, Baresel A, Roper M. Testability transformation. IEEE Trans. on Software Engineering, 2004,30(1):3−16.

附中文参考文献:

[1] Jorgensen PC.软件测试.第 2 版.北京:机械工业出版社,2003.5−7.

[15] 史忠植.知识发现.北京:清华大学出版社,2002.266−286.

薛云志(1979－),男,山西运城人,博士生,主要研究领域为软件测试,编译优化与测试.

陈伟(1977－),男,博士,主要研究领域为软件测试,并发系统建模与测试.

王永吉(1962－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为实时系统,网络优化,智能软件工程,优化理论,机器人,控制理论.

赵琛(1967－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为编译技术及应用,软件测试方法和工具.

王青(1964－),女,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为软件过程技术与质量管理.