对于测试用例生成的遗传算法改进①

吴 昊, 李浩然, 万交龙

(湖南大学 信息科学与工程学院, 长沙 410082)

摘 要: 软件测试技术中, 高效的测试用例生成能够大幅简化测试工作, 提高测试效率, 节省软件开发成本. 遗传算法作为一种高效的搜索寻优算法已被广泛应用到测试用例自动生成的研究中, 然而传统的遗传算法虽然具有良好的全局搜索能力, 但对于局部空间的求精问题却不是很有效, 存在早熟问题. 针对这些问题, 结合禁忌搜索算法, 对传统的遗传算法在适应度函数、遗传算子方面进行改进, 并进行遗传导向控制, 能够有效控制遗传早熟问题, 提高遗传算法的局部寻优能力. 实验结果表明, 本文所建议的方法在测试用例生成的效率和效果方面均优于基于传统遗传算法的测试用例方法.

关键词: 软件工程; 软件测试; 遗传算法; 禁忌算法; 测试用例生成

Improved Genetic Algorithm Used in Test Cases

WU Hao, LI Hao-Ran, WAN Jiao-Long

(School of Information Science and Engineering, Hunan University, Changsha 410082, China)

Abstract: In software testing process, efficient test case generation can dramatically simplify testing, improve test efficiency and save software development costs. As an effective search algorithm, genetic algorithm has been widely applied to the study on automatic generation of test cases, and has good global search capability. However, some inherent limits of this algorithm exist, such as low optimization efficiency, premature convergence, etc. This paper proposes a modified genetic algorithm combined with tabu search algorithm, improves the select and crossover operator of genetic algorithm against the shortcomings of premature convergence, and adopt the optimal preservation strategy for improving search capabilities in the local space and the overall operating efficiency. Experiments result shows that the new algorithm has obvious advantages in efficiency and effectiveness compared with traditional genetic algorithm for test case generation.

Key words: software engineering; software testing; genetic algorithm; tabu algorithm; test case generation

随着计算机的普及和信息化时代的到来, 应用软件的使用已经渗透到我们日常生活的各个方面, 比如网上购物、地图导航、手机支付、新闻浏览等等. 要使这些软件能够正常、有效的为我们服务, 我们必须保证它们的可靠性. 而软件测试是一种保证软件可靠性的重要方式, 它能够及时发现软件中存在的错误, 提高软件的整体质量. 在实际开发中, 花费在软件测试中的人力、物力和时间往往远远超过系统的编码实现, 在整个软件开发中占有极其重要的地位. 这是因为选取合适的软件测试用例(test case, TC)过程是非常复杂的, 它需要软件测试人员能够理解软件设计和执行的流程, 分析代码和判断临界条件, 具备较高的专业水准. 据相关统计, 在软件测试的全部开销中, 约40%花费消耗在设计 TC 阶段[1]. 除此之外, 单纯依靠手工完成测试样例的生成往往带有很大的盲目性, 导致 TC 生成的数量多, 测试效果差. 因此, 如何科学有效地依据软件规格或程序结构自动构造和生成 TC 变得极为必要, 已经成为软件测试的重点研究之一.

① 收稿时间:2015-12-23;收到修改稿时间:2016-01-29 [doi:10.15888/j.cnki.csa.005307]

© 中国科学院软件研究所 http://www.c-s-a.org.cn

软件测试样例的自动生成不仅能够显著提高测试的效率和性能, 而且能够极大地降低测试成本. 对于多结构的测试样例生成问题转为面向路径测试样例生成问题的解决尤甚, 更为重要的是, 基于路径的测试能够检测到软件程序中高达 65%的出错情况. 多年来, 在众多国内外研究机构的钻研和推动下, 许多新的技术和工具(如遗传算法[2]、退火算法[3]、免疫算法[4])已经被运用到软件测试样例的自动生成中, 并取得了丰硕的研究成果[5-10]. 通常的 TC 自动生成技术或基于随机, 或基于功能、结构以及出错条件, 其中基于结构的测试用例自动生成被最广泛的研究. 它通常被分为两类: 一类是面向路径的, 它利用输入的数据集来测试每条路径; 另一类是面向目标的, 它利用输入数据来测试说明, 而罔顾路径的选择, 典型的是数据流图测试.

遗传算法最初是在 1995 年由 Jones[2]首次在软件结构性测试中引入, 他的实验结果表明能够利用遗传算法解决测试用例自动生成问题[11], 并且只要选取合适的适应度函数就能够遍历每个判断分支, 并且所需要的测试用例数远小于随机算法. 虽然遗传算法在早期迭代阶段能够有很好的表现, 能在 TC 种群中生成大量次优解, 这些次优解能够在交叉算子的作用下高效的进化为最优解, 但是随着TC 群体迭代次数增加, 种群多样性的降低, TC 种群进化出最优解的速度将变慢, 甚至出现早熟现象, 影响了算法的性能.

为了避免早熟, 加快收敛速度, 生成高效、理想的测试用例, 本文提出一种结合禁忌搜索算法的改进遗传算法局部搜索能力的新方法. 该方法利用禁忌搜索算法能透过历史资料的维护来进行探索, 以克服只能寻求到局部最优解的情形, 同时不断扩大搜索空间, 最终达到全局最优解. 除此之外, 在种群进化过程中, 调整遗传算子使之具有自适应性, 通过使用禁忌搜索算法能够将种群中接近最优解和最优解的个体保存下来, 能够极大地减少进化过程中不收敛情况, 提高最优解搜索、进化速度, 增强遗传算法的局部搜索能力, 以及强化全局搜索能力.

1 基本理论

1.1 遗传算法在软件测试用例生成中的技术方法

遗传算法(genetic algorithm, GA)是一种高效的搜索寻优算法, 能够有效解决大空间、多峰值、非线性、全局化等高复杂度问题. 顾名思义, 遗传算法是基于自然界生物基因的遗传过程与达尔文的进化论中的适者生存而提出的一种算法. 算法的核心过程包括: a)种群染色体的编码和生成; b)适应度函数的设计和适应值计算; c)亲代选择策略; d)交叉/变异策略.

遗传算法的操作对象为染色体, 而染色体是由一连串基因组成. 将遗传算法引入到软件测试用例生成领域, 需要对被测程序的参数进行某种编码, 形成染色体. 而一系列染色体构成的集合称之为种群. 种群中的每个个体都有一个适应度值, 遗传算法通过适应度值来决定下一代生存的可能性. 下一代产生之后, 一部分个体的染色体进行交叉重组, 而少部分染色体的某个基因还可能发生变异, 从而进化到更优的种群, 为软件测试提供更高效的测试用例. 遗传算法通过对染色体的遗传控制来实现对最优解得求解.

在最开始阶段, 算法随机产生一个初始种群, 并且根据预先定义的适应规则, 计算每个种群成员的适应度值, 然后通过模拟自然界中的进化(选择、交叉、变异) 与淘汰机制, 对最开始的种群进行优胜劣汰和遗传操作, 产生新的一代种群作为新的测试集, 然后反复迭代使用进化规则和优胜劣汰规则, 直到满足条件, 生成符合要求的测试集, 退出循环. 基于遗传的 TC 自动生成算法其基本算法流程如图 1 所示.

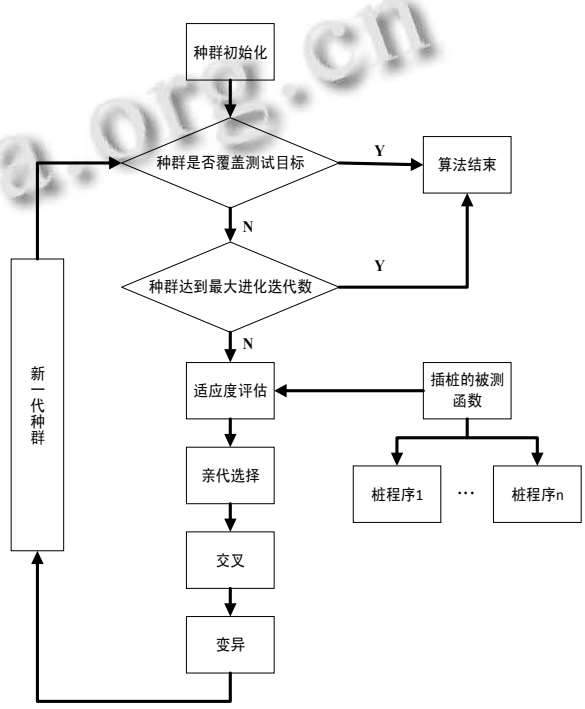

图 1 遗传算法流程

© 中国科学院软件研究所 http://www.c-s-a.org.cn

1.2 基于路径的测试用例生成

路径测试是普遍采用的软件测试方法之一, 它是利用输入的测试用例路径与预先设定的目标路径之间的匹配程度来评价测试用例的优劣. 利用基于路径的方法来自动生成测试用例时, 输入变量是程序的输入参数, 输出变量是程序运行到代码中某个位置时的一个或多个变量值(或者运行结果), 写成函数形式时, 待测程序可以表示为 $y = f(x)$, 其中 x 为输入参数(构成输入空间 V), y 为输出结果, f 为映射关系. 程序结构用控制流图表示为 $G = (V, E, s, e)$. 其中: V 表示语句节点; E 表示语句间可能存在的控制流向; s 和 e 分别对应程序的开始语句和结束语句. 若存在有序序列〈$V_0, ..., V_i, ..., V_n$〉, 其中, $V_i \in V$, 并且 V_{i+1} 节点可以在 V_i 节点后立即执行, 则该序列组成程序 G 的一条路径[12].

在测试用例生成过程中, 首先随机获取 G 中的一条路径 PT 作为目标路径, 然后使用遗传算法或者其他人工智能方法求取 $X \in D$, 使得 G 以 X 为输入时得到执行路径 PE. 当 PE 与 PT 重合时, X 即为所生成的测试用例.

2 改进遗传算法研究

2.1 染色体的编码和初始种群的构建

染色体的编码是遗传算法第一个要解决的问题, 选择合适的编码方式有利于提高算法的效率, 加快解得求解过程. 遗传算法的染色体编码是把一个问题的可行解从其解空间转换到遗传算法所能处理的搜索空间的转换方法. 与之对应的, 将遗传算法解空间向问题空间的转换则称为解码.

常用的染色体编码方法大致可以分为浮点数(实数)编码、最小字符集编码和符号编码这么几类. 由于二进制编码符合最小字符集原则, 便于用模式定理分析, 因此本文采用二进制编码作为染色体的编码方式. 而初始种群的构建, 则采用随机的方式在预先定义的阈值内进行随机产生.

2.2 适应度函数

适应度函数是用来评价在一个具体应用环境中种群中各个个体在该情形下的适应能力. 它的选定将直接影响到遗传算法解决问题的效率, 以及决定是否能够找出满足所需路径覆盖的测试用例. 常用的方法是通过对种群中个体对程序中分支覆盖或者路径覆盖情况来计算适应度值. 本文采用了 *Tracey*[13]所提出的"分支函数"插装的做法. 这种方法是预先在程序单元内部指定的逻辑路径所经过的每个分支点前插入一个实值函数 $f(x_1,x_2, ...,x_n)$(其中 $x_1, x_2, ..., x_n$ 为被测试单元的形参变量). 这些 f_i 用来评价被测单元的实际执行路径与指定逻辑路径之间的偏离程度. 对于分支函数 f_i, 它的取值取决于分支的选定. f_i 的具体选取方法如下:

1) 当分支判断为假时, f_i 的值为正;

2) 当分支判断为真时, f_i 的值为零, 这样, f_i 就是程序输入的一个函数.

当被测单元执行一组测试用例时, 被插桩在这些程序内部的实值函数将被计算出来. 因为这些分支函数 f_i 的值能够反映路径分支的覆盖情况, 因此可以将评价函数定义为所有 f_i 的联合表达形式, 即

$$F = F[f_1(x_1,x_2,...,x_n), f_x(x_1,x_2,...,x_n),...,f_n(x_1,x_2,...,x_n)]$$

这样, 评价函数 F 可归结为被测单元的形参变量 x_1 , x_2 , ..., x_n 的函数, 其值便成为评价形参变量的实际值(生成的测试用例)的优劣的一个很好的尺度.

关于 F 应具体采取何种形式,在本文中选用了不同种群个体间的分支函数的汉明距离作为评价标准, 即种群不同个体间遍历路径不同, 距离越大, 适应性越好. 每个个体适应度 F 值的计算形式为:

$$F_i = Hamming(f_i \oplus P_i) \quad (1)$$

式中 F_i 表示每个个体的适应度值, 值越大, 适应性越好, f_i 为每个个体的分支覆盖, P_i 为当前的路径集合. 具体计算步骤如下:

a) 初始化路径集合 P 为空

b) 判断该种群是否已经遍历完成, 如遍历完成, 则执行 e), 否则执行 c)

c) 若当前个体所经历的所有分支路径均已存在于集合 P 中, 则设置其适应度 F 为一个很小值, 执行 b). 否则, 执行 d)

d) 计算当前个体所经历分支路径与路径集合 P 的汉明距离作为该个体的适应度 F 值, 并更新路径集合 P, 使之存有当前已遍历所有个体所经历的路径, 然后执行步骤 b)

e) 算法结束, 输出每个个体的适应度值

由于种群中各个个体的 F 值有时可能相对相差很少, 从而导致各个个体的选择概率差别很小, 此时各个个体被选择的几率几乎一样, 这将导致遗传算法的选择功能被弱化, 需要对个体的 F 值进行标定(也就是

© 中国科学院软件研究所 http://www.c-s-a.org.cn

进行变换). 本文采用动态线性标定方法进行标定, 变换公式如下:

$$F' = F - F_{\min}^{k} - \xi^{k} \tag{2}$$

上式中 F'为变换后的适应值, F 为原适应值, $F_{\min}^{k}$ 为第 k 代个体的最小的适应值, ξ^{k} 为选择压力调节值, 它是一个较小的数, 它随着 k 的增大而减少, 一般采用如下的设置方法[14](本文选择 M 为 0.5, c 为 0.9):

$$\begin{cases} \xi^{0} = M \\ \xi^{k} = c \cdot \xi^{k-1} \end{cases}, \text{其中} M, c \text{为常数}, c \in [0.9 - 0.999] \tag{3}$$

2.3 遗传算子的选择与改进

在遗传算法中, 交叉概率和变异概率两个参数选择的优劣将极大地影响遗传算法的行为和性能. 针对不同的优化问题, 需要反复试验来确定交叉概率和变异概率, 一般很难找到适应于每个问题的最佳值. 因此, 如何选择合适的交叉概率和变异概率显得至关重要.

交叉概率指的是随机交叉亲代之间的等位基因的概率大小. 交叉概率越大, 种群能够越快产生新的个体. 然而, 过大的交叉概率会导致遗传模式被破坏的可能性也越大, 这会使已经具有高适应度的个体结构很快被破坏, 不利于优良个体进行代间传递; 但是如果交叉概率过小, 又会导致搜索过程变得非常缓慢, 不利于算法收敛.

变异概率指的是在产生新的个体的过程中, 亲代基因发生突变的概率大小. 在考虑如何避免种群早熟的情况下, 通常采用的是大变异概率的方法. 然而对于变异概率来说, 虽然大的变异概率能够不断更新种群的多样性, 克服早熟问题, 但是也会破坏种群中已有的最优解, 变成纯粹的随机搜索算法, 不利于算法收敛, 影响算法性能.

为了在交叉、变异过程中能够保证已有优良个体的特性, 本文采用自适应的交叉概率和变异概率, 使之能够根据适应度自动改变. 对于适应度较高的种群个体, 为其分配较低的交叉概率和变异概率, 使该个体的优良特性能够得到保护, 传递到下一代; 而对于适应度较低的种群个体, 则为其分配较高的交叉概率和变异概率, 使该个体能够被迅速淘汰掉. 由于自适应算法既能保证好的个体进行代间传递, 又能及时淘汰不适应环境要求的个体, 因此能够提供相对某个解的最佳交叉概率和变异概率.

本文采用的遗传算法中, 对遗传算子交叉概率 P_c 和变异概率 P_m 的自适应值计算公式如下[14]:

$$P_c = \begin{cases} \dfrac{k_1(F_{max} - F)}{F_{max} - F_{avg}}, F \geq F_{avg}, \\ k_2, F < F_{avg} \end{cases}$$

$$P_m = \begin{cases} \dfrac{k_3(F_{max} - F')}{F_{max} - F_{avg}}, F \geq F_{avg} \\ k_4, F' < F_{avg} \end{cases} \tag{4}$$

其中 F_{max}—群体中的最大适应值; F_{ave}—群体平均适应值; F—要交叉的两个个体中较大的适应值; F'—要变异个体的适应度值, k_1、k_2、k_3 和 k_4 为常数(本文分别取 0.5,0.9,0.02,0.09).

2.4 遗传导向控制

虽然在遗传操作过程中, 本文已经采用自适应算法对遗传算子进行控制, 来确保优良的种群个体能够尽量不被破坏, 传递到下一代, 但是还是具有一定的概率随机性. 为了进一步加快生成符合要求种群的效率, 需要对遗传操作进行控制.

这里以分支覆盖为例, 更好地描述这个问题. 假设某个种群个体 i 的分支覆盖情况为 v_i=[1 0 0 1 1 1 0 1 1 1], 而另一个种群个体 j 的分支覆盖情况为 v_j=[1 0 0 0 0 1 0 0 0 1]. 对比两个种群个体的分支覆盖情况, 可以发现个体 j 覆盖的分支结合恰好是个体 i 覆盖集合的一个子集. 虽然个体 i 和个体 j 都可能有很好的适应度值, 但是由于个体 i 覆盖的路径集合已经包含了个体 j 覆盖的路径集合, 若把它俩同时进行遗传, 就显得冗余, 没有必要, 可以根据具体情况和预先设定的参数值将个体 j 进行变异或者其他遗传操作. 因此在适应度评价中, 要将种群中个体适应度评价方法进行改进.

本文前面介绍的使用差分求个体适应度值的方法, 能够有效避免对于经历重复子路径的个体进行冗余遗传. 为了达到更好的遗传导向效果, 在以上方法的基础上还结合了禁忌搜索算法. 遗传时, 首先生成一个禁忌表, 该禁忌表的大小为当前种群覆盖测试路径率最高的最小子集的容量. 禁忌表中个体添加与每个个体的适应度的计算方法类似, 当不再有新的个体分支路径被添加到路径集合中时, 该禁忌表处于饱和状态. 同时, 设计了赦免准则(隔代赦免和饱和赦免)来提高搜索速度. 当全部邻域测试用例都被禁, 或者存在比当前全局最优测试用例更优的解被禁时, 按照赦免准则, 在禁忌测试用例的有效保持时间递减到零之前, 就可以将它们从 ST 中释放出来. 这样实现高效的全局最优搜索

很显然, 禁忌表中的个体为优先需要遗传的个体,

© 中国科学院软件研究所 http://www.c-s-a.org.cn

因为包含有最大的覆盖路径率. 对于剩下需要遗传的个体(控制种群大小比禁忌表容量要大), 为了使低概率的个体也有可能产生后代, 我们采用自适应法选择亲代进行交叉、变异. 这样不仅避免了最优解被污染, 而且可以把已有的好的特性遗传下去, 提高进化的效率和效果.

3 仿真实验

3.1 实验数据

为了验证本文改进遗传算法的有效性, 本文采用了三个典型待测程序[2](求余程序、三角形分类程序和查找程序)进行自动插桩, 及在每个分支语句处插入探针, 当输入数据运行待测程序时, 探针能够记录覆盖的分支情况, 并且保证插入的语句不影响待测程序本身.

3.2 实验环境和参数设置

本文所使用的实验环境和参数设置分别如表 1、表 2 所示.

表 1　实验环境设置

CPU	内存	操作系统
E6700 3.2GHz	2GB	Windows 7

表 2　算法参数设置

参数	设置值
种群大小 *NP*	10,20,50,**100**,150,200
遗传的最大代数 *NG*	10000
交叉概率 *Pc*	0.9
变异概率 *Pm*	0.05
自变量下界 *a*	-200
自变量上界 *b*	200
自变量离散精度 *eps*	1

3.3 实验结果比较与分析

本文选择求余程序(Program Remainder)、三角形分类程序(Program Tritype)和查找程序(Program Find)进行对比测试, 比较基于传统遗传算法(SGA)和改进遗传算法(EPGA)下的测试样例生成.

实验首先通过平均进化代数(ANEG)、成功率(RS, 即在1000次迭代之前停止)和首次失效发现平均TC数(F-measure, 即 FM)三个指标进行评估, 比较满足分支覆盖率为 100%情况下, 两种算法的性能. 通过MATLAB 工具进行仿真, 实验结果如表 3、表 4 和表 5 所示.

表 3　求余程序测试的实验结果

	Program Remainder		
	ANEG	RS	FM
SGA	61.3	1.0	276.4
EPGA	4.6	1.0	20.7

表 4　三角形分类程序测试的实验结果

	Program Tritype		
	ANEG	RS	FM
SGA	61.3	1.0	276.4
EPGA	4.6	1.0	20.7

表 5　查找程序测试的实验结果

	Program Find		
	ANEG	RS	FM
SGA	614.1	0.4	55318.4
EPGA	406.8	0.6	36632.3

通过比较表 3、表 4 和表 5 中两种方法在三种不同待测程序下的实验结果, 我们可以看出改进的遗传算法相对于之前的遗传算法性能有很大的提高.

再来看两种算法在不同种群规模下的迭代曲线, 如图 2 所示. 从图中我们可以看出, 使用改进的遗传算法生成覆盖所有分支的测试用例需要的迭代次数更少, 具有更好的迭代特性.

为了进一步验证改进算法的性能, 本文记录插桩三角形分类程序在种群规模为 100, 最大迭代次数为1000的条件下, 运行10次分别找到最优解的迭代次数和运行时间, 如表6所示(表格中"-"表示在该次算法执行中没有找到最优解). 从表中可以看出, 改进遗传算法不仅生成覆盖全路径的测试用例需要的时间更少, 而且收敛特性远远好于标准遗传算法(改进遗传算法10 次均收敛, 每次成功找到最优解; 而标准遗传算法只有两次能够找到最优解, 并且运行效率还不如改进算法), 有效的避免了早熟和不收敛问题, 验证了本文提出方法的有效性.

表 6　改进遗传算法和标准遗传算法对照

试验次数	改进遗传算法		标准遗传算法	
	迭代次数	运行时间	迭代次数	运行时间
1	251	4.9226	-	-
2	63	1.1615	140	2.6911
3	150	2.8368	-	-
4	267	5.0427	-	-
5	281	5.2913	-	-
6	320	6.4575	-	-
7	38	0.80789	93	1.8531
8	339	6.8389	-	-
9	447	9.3519	-	-
10	235	4.6247	-	-

© 中国科学院软件研究所 http://www.c-s-a.org.cn

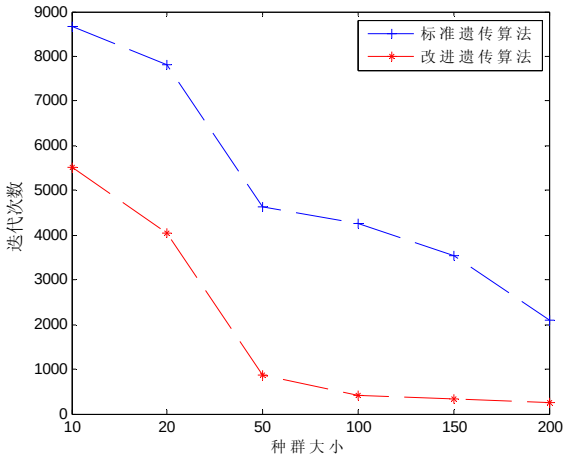

图 2 两种算法在不同种群大小下的迭次次数对比图

4 结语

TC 自动生成算法的研究大大提高了软件测试的自动化水平, 为软件失效数据采集提供了一种自动化方法. 然而传统遗传算法存在着局部搜索能力差、早熟等缺点, 往往早早地找到了次优解, 从次优解到最优解的寻找过程, 却需要付出很大的代价. 为了改进遗传算法的这些缺点, 本文改进了遗传算法的适应度函数, 实现最优保存策略, 结合禁忌搜索算法, 大大改善了遗传算法的局部求精能力, 同时使交叉、变异算子具有自适应性, 避免早熟, 从而提高了遗传算法的运行效率和效果.

参考文献

1 聂鹏,耿技,秦志光,软件测试用例自动生成算法综述.计算机应用研究,2012,(2):401–405,413.

2 Jones BF, Sthamer HH, Eyres DE. Automatic structural testing using genetic algorithms. Software Engineering Journal, 1996, 11(5): 299–306.

3 Burnim J, Sen K. Heuristics for Scalable Dynamic Test Generation. Proc. of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering. 2008, IEEE Computer Society. 443–446.

4 Liaskos K, Roper M. Automatic test–data generation: An immunological approach. Testing: Academic and Industrial Conference Practice and Research Techniques-MUTATION, 2007. TAICPART-MUTATION 2007. 2007. IEEE.

5 Godefroid P, Klarlund N, Sen K. DART: Directed automated random testing. ACM Sigplan Notices. 2005. ACM.

6 Estero-Botaro A, et al. A framework for genetic test-case generation for WS-BPEL compositions. Testing Software and Systems. Springer. 2014: 1–16.

7 Gen M, Lin L. Multiobjective evolutionary algorithm for manufacturing scheduling problems: State-of-the-art survey. Journal of Intelligent Manufacturing, 2014, 25(5): 849–866.

8 Zhou Y, Sugihara T, Sato Y. Applying GA with tabu list for automatically generating test cases based on formal specification. Structured Object-Oriented Formal Language and Method. Springer. 2014: 17–31.

9 Li K, Zhang Z, Liu W. Automatic test data generation based on ant colony optimization. 2009 Fifth International Conference on Natural Computation. 2009. IEEE.

10 Shen X, et al. Automatic generation of test case based on GATS algorithm. IEEE International Conference on Granular Computing(GRC'09). IEEE. 2009.

11 荚伟,奚红宇,高仲仪,遗传算法在软件测试数据生成中的应用.北京航空航天大学学报,1998,(4):68–71.

12 Ahmed MA, Hermadi I. GA-based multiple paths test data generator. Computers & Operations Research, 2008, 35(10): 3107–3124.

13 Tracey N, Clark JA, Mander K. The way forward for unifying dynamic test-case generation: The optimisation-based approach. Proc. of the IFIP International Workshop on Dependable Computing and Its Applications (DCIA). York. 1998.

14 龚纯,王正林,精通 MATLAB 最优化计算.北京:电子工业出版社,2009.

© 中国科学院软件研究所 http://www.c-s-a.org.cn