

- 然后,根据每个目标关键点路径与进化个体覆盖的关键点路径之间的距离计算个体针对每一条目标路径的适应度函数值分量;
- 检查个体是否覆盖了某一目标路径:如果是,则保存测试数据及其覆盖的路径,并简化多个目标的优化问题;如果不是,则采用遗传操作以生成新一代种群.不断迭代,直到遗传算法满足终止条件.

这里的适应度函数由若干个基于难覆盖路径的适应度函数分量构成,即,算法对种群中的每条个体,逐个检测其是否满足某个适应度函数分量:如果满足,则保留该个体为测试数据,并删除该目标对应的适应度函数分量,直到所有的适应度函数分量都有个体满足时算法成功.

4.2.1 启发式信息

利用已确定的易覆盖关键点路径及其测试数据提供的信息,生成难覆盖路径的测试数据.可从两方面理解:

- (1) 从覆盖所有关键点角度考虑.将关键点图中的各关键点记为 $K=\{k_1,k_2,k_3,\dots,k_{i-1},k_i,k_{i+1},\dots,k_n\}$,其中,已经确定了覆盖关键点 $k_1,k_2,\dots,k_{i-1},k_{i+1},\dots,k_n$ 的测试数据,但还未找到覆盖关键点 k_i 的测试数据,则考虑利用覆盖 k_{i-1} 和 k_{i+1} 的两组测试数据的启发式信息进行搜索;
- (2) 从覆盖关键点组合路径的角度考虑.例如,难覆盖的关键点路径为 $P:entry-2-a'-b'-c-exit$;已找到覆盖关键点路径 $Q:entry-2-a'-b'-c'-exit$ 的测试数据,则考虑利用覆盖路径 Q 的测试数据提供的启发式信息进行搜索,其中, c 和 c' 是兄弟分支子关键点.

容易理解:如果一组测试数据覆盖了路径 Q 而没有覆盖路径 P ,且 P 和 Q 这两条路径仅有一个兄弟分支子关键点不同,则覆盖路径 P 的测试数据与覆盖路径 Q 的测试数据之间一定存在某种关系,即,是否覆盖的那个关键点的分支判断中涉及到的变量的取值不同.此时,必定可以在覆盖了路径 Q 的数据基础上发生变化,生成覆盖路径 P 的数据.在最理想的状态下,甚至只需微调覆盖 Q 路径测试数据的某个分量,便能得到理想的数据.

4.2.2 初始种群

初始种群由两部分构成:一是包含了启发式信息的个体,二是随机选择的个体.

- 包含启发式信息的个体:根据图 5 所示的快速软件测试数据自动生成模型,在分析被测程序之后,得到控制流图、关键点图,最终得到关键点表示的路径集 A ;
- 随机生成测试数据集:一条编码就是覆盖一条路径的测试数据.待测程序输入参数个数为 n ,编码长度为 n .以这些测试数据为输入,运行插装后的被测程序,此时,测试数据覆盖的是容易覆盖的路径,构成易覆盖路径集 E .

由 $A \sim E$ 得到 H' ,使用不可行路径检测模型检测 H' ,将 H' 分成两部分:难覆盖路径集 H 和不可行路径集 U ,即:

$$H'=H \cup U.$$

H 就是待覆盖的目标路径集.保留一组能够覆盖集合 E 中路径的测试数据作为最终测试数据集的一部分;在被保留的测试数据中,如果其覆盖的路径与难覆盖路径邻近,则保留该测试数据作为遗传算法初始种群中的个体,这些个体通过复制的方式占到初始种群中个体数目的一半.

初始种群中的另一半个体使用轮盘赌的方式,从初始时随机生成的测试数据个体中选定.

4.2.3 适应度函数

本文使用寻找多个目标的遗传算法^[8],其适应度函数包含两个层次:底层是个体对多个目标中每一个目标的满足程度;顶层是多个目标逐一被满足,直到所有的多目标都被满足.具体来说:

- 在顶层,适应度函数值由 m 维向量表示,向量中的每个分量代表个体针对某一条待覆盖路径的适应值, m 是待覆盖路径的数目.如果第 i 个个体使第 j 个适应度函数分量为 0,则该个体为覆盖第 j 条路径的测试数据,适应度函数中的第 j 个向量被删去,目标由原来的 m 个变成 $m-1$ 个,直至所有目标都被满足;
- 在底层,以进化个体 x 为例, x 为被测程序 G 的一组输入,计算个体针对第 k 条难覆盖路径 P_k 的适应度函数分量值 $f_k(x)$, $f_k(x)$ 由层接近度和分支距离这两部分组成^[2].

适应度函数的形式化表示如下^[7]:

$$f(x)=\min(f_1(x),f_2(x),\dots,f_m(x)) \quad (1)$$

$$f_k(x) = appr_{p_k}(x) + dist_{p_k}(x) \quad (2)$$

$$appr_{p_k}(x) = \frac{b(p_k, P(x))}{|p_k|} \quad (3)$$

$$dist_{p_k}(x) = \begin{cases} 0, & \text{满足分支条件} \\ |c_2 - c_1| + k, & \text{不满足分支条件} \end{cases} \quad (4)$$

其中, $P(x)$ 表示个体 x 覆盖的关键点路径; P_k 为待覆盖的第 K 条关键点路径; $b(P_k, P(x))$ 表示两条路径的层接近度,其值为从第 1 个关键点开始 P_k 与 $P(x)$ 的不同节点的数目; $|P_k|$ 是目标路径 P_k 中包含的关键点的数目; C_1 op C_2 为分支节点的表达式; k 是一个常数,用以表达 $C_2 \neq C_1$ 的情况.当个体 x 覆盖某条难覆盖的目标路径时,保留该个体,并将与该目标路径对应的目标函数分量删除,此时的适应度函数中的目标数减 1.即:随着难覆盖路径测试数据的逐一确定,目标函数的个数逐个减少,直至算法找到满足最后一个目标路径的测试数据.

4.2.4 算法步骤

使用遗传算法生成难覆盖的关键点路径测试数据的步骤如下.

- Step 1. 在定义域内随机生成测试数据集.
- Step 2. 使用第 4.2.2 节中的方法初始化种群.
- Step 3. 检查是否满足算法终止条件:待覆盖关键点路径为 0 或达到一定迭代次数.若是,则转到 Step 7.
- Step 4. 以个体为输入运行插装后的被测程序,计算个体适应度函数值.
- Step 5^[8]. 判断个体适应度向量值是否有 0 分量:如果有,保存该个体为测试数据,待覆盖目标路径个数减 1;如果该测试数据覆盖的关键点路径与尚未覆盖的关键点路径编号相邻,则保留该个体到下一代种群中.
- Step 6. 遗传操作,使用选择、交叉和变异算子,结合第 4.2.2 节中的方法产生新一代种群,转到 Step 3.
- Step 7. 停止进化,输出测试数据.

5 仿真实验

选择 3 个基准程序验证所提方法的有效性,将本文方法(F -method)与 Ahmed 方法^[7]、单目标路径方法(single target method)^[4]、多路径覆盖方法(multi-targets method)^[8]及 N 本文方法(NF -method)在同样的被测程序上实验.其中,“ N -本文方法”是指使用本文快速生成测试数据的方法,但在遗传算法优化过程中不使用启发式信息.所有实验程序均采用 Matlab 语言编写,并在 Matlab 2012 中运行.微机配置为 Windows(Intel(R) Core(TM)2 Duo CPU E8200,2.66GHz,2.00GBRAM,32 位操作系统.与遗传算法相关的策略及参数设置见表 1.为减少随机因素的影响,针对不同程序每种情况独立运行 100 次,以生成覆盖目标路径的测试数据或达到最大进化代数作为算法运行的终止条件.实验记录中的时间为进化 100 次的总时间.

Table 1 Parameters in the genetic algorithm

表 1 遗传算法中的参数设置

遗传算法策略(参数)	取值
选择策略	精英选择,轮盘赌选择
交叉策略	单点随机交叉
交叉概率	0.9
变异策略	单点变异
变异概率	0.3
最大迭代次数	1 000
种群大小	30

表 2 说明了 3 个基准程序的信息.

Table 2 Basic information of the programs

表 2 被测程序的基本信息

程序	分支结构	输入分量个数	关键点理论路径数	可行路径数	不可行路径数	定义域
三角形	3层选择嵌套	3	4	4	0	$[0,100]^3$
3个数排序	选择并列关系	3	8	7	1	$[0,100]^3$
冒泡程序	2层循环嵌套,内层嵌套中有1个选择	3	8	7	1	$[0,100]^3$

表 3 说明本文方法的两大优势:

- (1) 测试数据生成效率;
- (2) 启发式信息对遗传算法优化效率的影响.

对于三角形分类程序, F -method 仅需对 1 条难覆盖路径构造适应度函数,即:适应度函数中只有一个目标分量,其生成覆盖目标路径测试数据所需的进化代数和时间显著优于 Ahmed 方法和 Single target.由于 F -method 需要在 GA 运行前确定易覆盖路径的测试数据,所以在平均运行时间上略优于 Multi-targets.对于 3 个数排序和冒泡程序,本文所提方法也优于对比的方法.事实上, F -method 的优势不仅在于算法运行时效率的提高,更在于节省了算法运行前的工作量.从适应度函数设计的工作量来看,其他 3 种方法都需要设计所有待覆盖路径的适应度函数,而 F -method 仅需为难覆盖路径设计适应度函数.甚至于,在本文方法初始时随机生成的测试数据中,还有可能一次即随机生成覆盖所有可行路径的测试数据,无需进入遗传算法的迭代过程,并且可以快速判定出不可行路径的数目及具体路径.将“ NF -method”与 F -method 的结果对比, NF -method 中由于初始种群中不含启发式信息,其平均进化代数和平均时间都差于 F -method,但仍显著优于 Ahmed 和 Single target,略优于 Multi-targets.这是由于,在 Multi-targets 方法中,在算法初期即能生成覆盖易覆盖路径的测试数据,其多出的时间消耗主要在于对各目标路径的适应度函数计算上,寻找难覆盖路径测试数据的时间与 NF -method 大致相当,且二者都需要逐个检查种群中的个体是否为所需的测试数据.事实上, NF -method 和 F -method 显著优于 Multi-targets 方法的优势不在于进化代数和进化时间,而在于适应度函数的设计和计算.

Table 3 Performance index value by using different methods in three benchmark programs

表 3 3 个基准程序不同测试数据生成方法的性能指标值

程序	方法	适应度函数个数	时间(s)	运行时间率(%)	平均迭代次数	迭代率(%)
三角形	Ahmed 方法	4	0.293 6	18.87	63.7	35.47
	单目标路径方法	$1 \times 4 = 4$	0.437 8	12.65	373.3	6.05
	多路径覆盖方法	4	0.097 0	57.11	43.1	52.43
	N -本文方法	1	0.090 5	61.21	42.5	53.17
	本文方法	1	0.055 4	100	22.6	100
3 个数排序	Ahmed 方法	7	0.928	30.097	46.9	31.34
	单目标路径方法	$1 \times 7 = 7$	4.68	5.97	482.5	3.04
	多路径覆盖方法	7	0.467	59.80	23.8	61.76
	N -本文方法	1	0.452	61.79	21.5	68.37
	本文方法	1	0.279 3	100	14.7	100
冒泡程序	Ahmed 方法	7	0.321	33.95	40.2	25.07
	单目标路径方法	$1 \times 7 = 7$	1.764 2	6.18	230.4	4.37
	多路径覆盖方法	7	0.245 4	44.41	32.0	31.5
	N -本文方法	1	0.214 9	50.72	25.6	39.375
	本文方法	1	0.109 0	100	10.08	100

表 4 是对本文所提方法的自身性能分析,着重从工作量角度考虑.随机生成所有可行路径的次数指的是在 100 次的重复实验中,无需遗传算法迭代,在随机生成的测试数据中就能找到覆盖所有可行路径的测试数据的成功次数.可以这样理解: F -method 充分利用了随机算法快速生成测试数据的优势,此时,随机生成的测试数据即覆盖了所有可覆盖的路径;或者可以理解为:本次算法的执行没有产生难覆盖的路径,已经随机生成了所有需要的测试数据. GA 表示 100 次实验中需要遗传算法优化以生成“难覆盖路径”的测试数据的次数.虽然随机算法

难以生成所有的测试数据,如在三角形分类程序中,100次的程序运行只有18次覆盖到全部待测路径,但其找到难覆盖路径的准确率达到了100%,且一次生成了75%以上的测试数据,在3个数排序和冒泡程序中,一次随机生成了80%的测试数据,这使得后继的遗传算法优化过程中的适应度函数设计和计算的工作量大为减少,而已获得的测试数据也为算法优化效率的提高提供了启发信息。

Table 4 Performance analysis of the *F*-method

表 4 本文方法(*F*-method)性能指标分析

程序	理论 路径数	易覆盖关键点路径			难覆盖关键点路径			不可行路径	
		条数	随机生成所有 可行路径的次数	遗传 算法	条数	成功率(%)	时间	实际 条数	找到
三角形	4	3	18	82	1	100	0.059 8	0	0
3个数排序	8	6	22	78	1	100	0.041 2	1	1
冒泡程序	8	6	19	81	1	100	0.020 7	1	1

表 4 中的“难覆盖关键点路径时间”指的是运行遗传算法生成难覆盖路径测试数据的时间,而表 3 中的“时间”中还包含随机生成测试数据及确定难覆盖路径的时间.并非所有程序都存在不可行路径,程序的不可行路径也可能不止一条.本文实验中,3 个数排序程序和冒泡排序都存在 1 条不可行路径,使用本文方法能够快速、准确地判断 3 个基准测试程序的不可行路径。

工业程序 Flex 和 Sed 广泛用于验证不同测试数据生成方法的性能.程序的源代码可以从系统“Software-artifact Infrastructure Repository”下载^[18].对于每个被测程序,各选取 1 个函数进行实验,该函数的结构、输入向量包含的分量个数、理论路径数、可行路径数和代码行数见表 5.所有程序运行 30 次,取平均值.遗传算法种群大小为 500,算法终止条件为找到全部解或达到指定迭代次数.其他参数见表 1.

Table 5 Basic information of the industry programs

表 5 被测工业程序的基本信息

程序	分支结构	输入 变量数	关键点 路径数	可行 路径数	不可行 路径数	代码 行数
Flex	1 个 if 选择和 1 个循环,该循环中嵌套 1 个 if-else 选择	22	32	27	5	107
Sed	1 个循环,其中嵌套 1 个 if-else 选择	11	16	16	0	54

如表 5 和表 6 所示:对于 Flex 程序,使用随机方法未能生成覆盖所有可行路径的测试数据,在限定迭代次数为 100 的前提下,找到难覆盖路径测试数据的机率是 76.7%;Sed 程序在初始种群大小为 500 的前提下,每次找到的难覆盖路径不完全相同,平均难覆盖路径数为 1.8.当算法初始种群增大时,难覆盖路径数目相应减少,在 30 次随机生成测试数据的实验中,有 6 次生成了覆盖全部可行路径的测试数据。

Table 6 Performance analysis of the *F*-method in industry programs

表 6 工业程序中本文方法(*F*-method)性能指标分析

程序	理论路径 数目	易覆盖的关键点路径			难覆盖的关键点路径			不可行路径	
		条数	随机生成所有 可行路径的次数	遗传 算法	条数	成功	成功率(%)	实际 条数	找到
Flex	32	24	0	30	3	23	76.7	5	5
Sed	16	14	6	24	1.8	30	100	0	0

表 7 中程序以找到所有测试数据为终止条件,Flex 的平均迭代次数为 72.6,平均运行时间为 6.984s;而 Sed 程序仅需 32.5 次迭代和 0.188s.这是由于,Flex 程序的理论目标路径数目和输入变量数都较 Sed 要多,且其分支结构不同,Sed 仅循环体内存在一个选择分支,而 Flex 程序中循环体内外都有选择分支.在生成难覆盖路径测试数据的过程中,两个被测程序优化算法的迭代率相差较大.这是由于,Flex 函数有更多的输入变量,其覆盖不同路径的测试数据间提供的启发式信息的价值相对较小,算法需要更多次的迭代.对比 *NF*-method 与 *Multi-targets* 算法在两个程序上的优化效果,迭代次数相差仅为 1.57 和 1.3,迭代时间也仅相差 0.648s 和 0.033s.这是由于,对

两种算法而言,运行时间主要集中于寻找难覆盖路径的测试数据.*NF-method* 比 *Multi-targets* 方法的微弱优势仍在于其对适应度函数的设计和计算上.*F-method* 在时间和迭代次数上显著优于 *NF-method* 和 *Multi-targets* 算法,主要是因为易覆盖路径测试数据对 *F-method* 所提供的启发式信息.除此之外,*NF-method* 和 *F-method* 在算法运行前仅需设计极少的适应度函数,这一部分节省的时间和工作量没有体现在表 6 中.

Table 7 Performance index value by using different methods in industry programs

表 7 工业程序不同测试数据生成方法的性能指标值

程序	方法	适应度函数个数	时间(s)	运行时间率(%)	平均迭代次数	迭代率(%)
Flex	多路径覆盖方法	32	8.961	77.93	93.57	77.58
	<i>N</i> -本文方法	3	8.313	84.01	92	78.91
	本文方法	3	6.984	100	72.6	100
Sed	多路径覆盖方法	16	0.412	45.63	79.5	40.88
	<i>N</i> -本文方法	2	0.379	49.60	78.2	41.56
	本文方法	2	0.188	100	32.5	100

上述实验结果表明,本文方法在得到覆盖目标路径的测试数据方面具有显著优势.与 *Ahmed* 方法、单目标路径方法、多目标路径生成方法及不使用启发信息的本文方法相比,本文所提方法使用较少的工作量快速地生成了测试数据;同时还确定了待测程序的不可行路径.

6 性能分析

本文的主要贡献在于:(1) 提出一种基于关键点的路径表示法,能够计算出程序的理论路径条数,并能够快速确定已知路径的邻近路径;(2) 建立了软件测试数据快速自动生成方法的完整模型,为使用遗传算法高效地生成路径覆盖的测试数据提供了求解的框架,实现了测试数据和不可行路径的快速确定,极大地减少了算法中适应度函数设计和计算的工作量.其具体优势在于:

- 1) 具有工程意义:从软件测试工作的整体效率出发,提出测试数据快速生成的完整模型,一次实现测试数据的生成和不可行路径的判断,且减少了算法中适应度函数设计和计算的工作量,极大地提高了测试数据生成的效率;
- 2) 测试数据生成效率高:
 - (1) 优化目标少:在以往的方法中,算法寻找覆盖指定路径的测试数据,被测程序的所有可行路径都是待覆盖的目标.例如在多路径覆盖测试数据进化过程中,有 m 条路径就有 m 个初始目标.本文的测试数据快速生成方法一次确定被测程序的绝大部分测试数据,明确难覆盖路径和不可行路径,优化目标大为减少;
 - (2) 启发式信息:利用已有测试数据生成难覆盖路径的测试数据.由于关键点路径表达式只由分支子关键点表示,容易确定难覆盖路径其前和其后的易覆盖关键点路径及其测试数据,将这些测试数据作为优化算法初始种群的一部分,在此基础上进行优化,能够较快地找到覆盖难覆盖路径的测试数据;
- 3) 工作量的减少:本文方法中使用算法寻找的仅是难覆盖路径的测试数据,算法优化的目标(难覆盖路径)数目大为减少,相应的适应度函数的设计和计算大量减少,也在一定程度上降低了优化难度.而且,判断关键点路径是否被覆盖时使用的程序插装变量和语句,因分支子关键点的成对出现而简化,也减少了一定的工作量;
- 4) 快速确定不可行路径:使用不可行路径检测模型对未被测试数据覆盖到的路径进行检测,以确定不可行路径,其余的路径是难覆盖路径.不可行路径检测模型检测小部分未被覆盖到的路径所需的工作量明显少于检测全部路径的工作量.

7 总 结

本文提出一种基于关键点路径表示的软件测试数据快速自动生成方法,所提出的关键点路径表达式能够计算程序的理论路径数目,分支子关键点成对出现的特点能够减少插装工作量,并根据已知路径快速确定其邻近路径;将在被测程序定义域内随机生成的测试数据运行于插装后的程序,确定被覆盖的路径及相应的测试数据;对未被覆盖的路径进行不可行路径的检测,从而将所有的关键点路径分为易覆盖路径、难覆盖路径和不可行路径3类;在遗传算法中,利用易覆盖路径及其测试数据提供的启发式信息,仅需设计和计算少量适应度函数,即可生成覆盖难覆盖路径的测试数据;由此给出了基于关键点路径表示的软件测试数据快速自动生成的完整模型.对3个基准程序和2个工业程序的实验结果表明,采用本方法生成测试数据具有更高的效率.在未来的工作中,将进一步思考易覆盖路径测试数据的生成方法以及如何更好地利用易覆盖路径的信息生成难覆盖路径的测试数据.

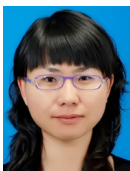
References:

- [1] Qiu XK, Li XD. A path-oriented tool supporting for testing. *Acta Electronica Sinica*, 2004,32(12A):231–234 (in Chinese with English abstract).
- [2] Gong DW, Yao XJ, Zhang Y. *Evolution Theory and Application for Testing Data Generation*. Beijing: Science Press, 2014. 8–32 (in Chinese).
- [3] Sthamer HH. The automatic generation of software test data using genetic algorithms [Ph.D. Thesis]. Pontyprid: University of Glamorgan, 1995.
- [4] Mansour N, Salame M. Data generation for path testing. *Software Quality Journal*, 2004,12(2):121–136. [doi: 10.1023/B:SQJO.0000024059.72478.4e]
- [5] Aldeida A, Lars G. Test data generation with a Kalman filter-based adaptive genetic. *The Journal of Systems and Software*, 2015, 103:343–352. [doi: 10.1016/j.jss.2014.11.035]
- [6] Jiang SJ, Shi JJ, Zhang YM, Han H. Automatic test data generation based on reduced adaptive particle swarm optimization algorithm. *Neurocomputing*, 2015,158:109–116. [doi: 10.1016/j.neucom.2015.01.062]
- [7] Ahmed MA, Hermadi I. GA-Based multiple paths test data generator. *Computer & Operations Research*, 2008,35(10):3107–3124. [doi: 10.1016/j.cor.2007.01.012]
- [8] Gong DW, Zhang WQ, Zhang Y. Evolutionary generation of test data for multiple paths coverage. *Chinese Journal of Electronics*, 2011,19(2):233–237.
- [9] Ghiduk AS. Automatic generation of basis test paths using variable length genetic algorithm. *Information Processing Letters*, 2014, 114:304–316. [doi: 10.1016/j.ipl.2014.01.009]
- [10] Gong DW, Zhang Y. Novel evolutionary generation approach to test data for multiple paths coverage. *Acta Electronic Sinica*, 2010,38(6):1299–1304 (in Chinese with English abstract).
- [11] Paul CJ, Wrote; Han K, Du XT, Trans. *Software Testing*. 2nd ed., Beijing: Machine Press, 2003. 62–83 (in Chinese).
- [12] Genford JM, Wrote; Zhang XM, Huang L, Trans. *The Art of Software Testing*. 3rd ed., Beijing: Machine Press, 2012. 40–68 (in Chinese).
- [13] Long Q, Wu CZ, Huang TW, Wang XY. A genetic algorithm for unconstrained multi-objective optimization. *Swarm and Evolutionary Computation*, 2015,22:1–14. [doi: 10.1016/j.swevo.2015.01.002]
- [14] Zheng JN, Chien CF, Mitsuo G. Multi-Objective multi-population biased random-key genetic algorithm for the 3-D container loading problem. *Computers & Industrial Engineering*, 2014,89:80–87. [doi: 10.1016/j.cie.2014.07.012]
- [15] Ngoc M, Tan HBK. Heuristics-Based infeasible path detection for dynamic test data generation. *Information and Software Technology*, 2008,50(7-8):641–655. [doi: 10.1016/j.infsof.2007.06.006]
- [16] Gong DW, Yao XJ. Automatic detection of infeasible paths in software testing. *IET Software*, 2010,4(5):361–370. [doi: 10.1049/iet-sen.2009.0092]
- [17] Mickael D, Bernard B, Arnaud G. Infeasible path generalization in dynamic symbolic execution. *Information and Software Technology*, 2015,58:403–418. [doi: 10.1016/j.infsof.2014.07.012]

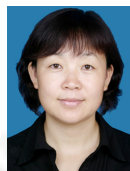
- [18] SIR. A repository of software-related artifacts meant to support rigorous controlled experimentation. <http://sir.unl.edu/portal/index.php>

附中文参考文献:

- [1] 邱晓康,李宣东.一个面向路径的软件测试辅助工具.电子学报,2004,32(12A):231-234.
[2] 巩敦卫,姚香娟,张岩.测试数据进化生成理论及应用.北京:科学出版社,2014.8-32.
[10] 巩敦卫,张岩.一种新的多路径覆盖测试数据进化生成方法.电子学报,2010,38(6):1299-1304.
[11] Paul CJ,著;韩柯,杜旭涛,译.软件测试.第2版,北京:机械工业出版社,2003.62-83.
[12] Genford JM,著;张晓明,黄琳,译.软件测试的艺术.第3版,北京:机械工业出版社,2012.40-68.



丁蕊(1977-),女,辽宁台安人,讲师,CCF 学生会员,主要研究领域为软件测试,演化算法.



张岩(1972-),女,博士,教授,CCF 会员,主要研究领域为基于搜索的软件工程.



董红斌(1963-),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为自然计算,机器学习,多 Agent 系统,数据挖掘.



冯宪彬(1972-),男,副教授,主要研究领域为软件测试,演化算法.