软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2016,27(3):593−610 [doi: 10.13328/j.cnki.jos.004983]

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

# 面向条件判定覆盖的线性拟合制导测试生成*

汤恩义[1,2], 周 岩[1,3], 欧建生[1,3], 陈 鑫[1,3]

[1](计算机软件新技术国家重点实验室(南京大学),江苏 南京 210023)

[2](南京大学 软件学院,江苏 南京 210093)

[3](南京大学 计算机科学与技术系,江苏 南京 210023)

通讯作者: 陈鑫, E-mail: chenxin@nju.edu.cn

**摘 要**: 条件判定覆盖(condition/decision coverage,简称 C/DC)准则是各种安全攸关软件测试中常用的测试覆盖准则,它要求软件测试覆盖程序中每个判定以及条件的真/假取值.现有的自动测试生成方法在针对该准则的测试用例生成过程中存在很多不足.例如:符号执行方法很难处理较为复杂的非线性条件约束,并在处理程序的规模上受到很大限制;希尔攀登法由于在搜索过程中易陷入局部最优,而难以达到满足 C/DC 准则的高覆盖率;模拟退火法和遗传算法依赖于用户使用过程中的复杂配置,测试用例生成效果具有一定的随机性.针对这一现状,提出了一种线性拟合制导测试用例生成方法.依据 C/DC 准则,该方法将程序中的每一个条件判定规范化为一个与零值比较的数值函数,并以插桩与执行获得该函数当前输入下的采样.通过拟合这些采样,能够逐步判断出程序中各个条件判定与输入的关系,并利用这些关系生成高覆盖率的测试用例.相对于传统方法,该方法具有参数配置简易、生成过程高效等优点,并且能够处理带非线性条件约束、逻辑复杂的程序.在 3 个开源软件库中的 25 个真实程序上运行的实验结果表明,所提出的方法比目前以覆盖率见长的遗传算法(genetic algorithm,简称 GA)制导方法具备更好的覆盖能力与更高的执行效率.

**关键词**: 测试用例自动生成;条件判定覆盖;线性拟合;关联路径

**中图法分类号**: TP311

中文引用格式: 汤恩义,周岩,欧建生,陈鑫.面向条件判定覆盖的线性拟合制导测试生成.软件学报,2016,27(3):593−610. http://www.jos.org.cn/1000-9825/4983.htm

英文引用格式: Tang EY, Zhou Y, Ou JS, Chen X. Test generation approach guided by linear fitting for condition/decision coverage criteria. Ruan Jian Xue Bao/Journal of Software, 2016,27(3):593−610 (in Chinese). http://www.jos.org.cn/1000-9825/4983.htm

## Test Generation Approach Guided by Linear Fitting for Condition/Decision Coverage Criteria

TANG En-Yi[1,2], ZHOU Yan[1,3], OU Jian-Sheng[1,3], CHEN Xin[1,3]

[1](State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210023, China)

[2](Software Institute, Nanjing University, Nanjing 210093, China)

[3](Department of Computer Science and Technology, Nanjing University, Nanjing 210023, China)

* 基金项目: 国家自然科学基金(61402222, 91318301, 61561146394); 国家重点基础研究发展计划(973)(2014CB340703); 教育部高等学校博士学科点专项科研基金(20110091120058); 江苏省产学研项目(BY2014126-03)

Foundation item: National Natural Science Foundation of China (61402222, 91318301, 61561146394); National Program on Key Basic ResearchProject of China (973 Program) (2014CB340703); Specialized Research Fund for the Doctoral Program of Higher Education (20110091120058); Project on the Integration of Industry, Education and Research of Jiangsu Province (BY2014126-03)

收稿时间: 2015-07-15; 修改时间: 2015-10-20; 采用时间: 2015-11-27; jos 在线出版时间: 2016-01-05

CNKI 网络优先出版: 2016-01-05 16:39:56, http://www.cnki.net/kcms/detail/11.2560.TP.20160105.1639.009.html

**Abstract**: Condition/decision coverage (C/DC) is a frequently used coverage criteria for safety-critical software testing. It requires every decision and condition in the program have taken all possible outcomes (true or false). Existing approaches of automatic test generation for C/DC criteria are defective. For example, symbolic execution based approaches are limited by the constraint solver, which is difficult in processing non-linear constraints; hill climbing often sticks at local optima, which limits yielding of high-coverage cases; and simulated annealing and genetic algorithm need complicated configuration, which make the results unstable. In this paper, a novel test generation approach that is guided by linear fitting is proposed. The basis of the approach is to sample every decision and condition of numerical values with program instrumentation. The relationship of inputs and samples is then build with linear fitting functions. By searching the target inputs on the gradually refined functions, test case is generated with high coverage. Experiments on 25 real programs in open source projects show that the proposed approach is more effective and efficient than the genetic algorithm of test generation

**Key words**: automatic test generation; condition/decision coverage; linear fitting; associated path

软件测试是软件开发过程中不可或缺的步骤之一,也是工业界保障软件质量最主要的手段.在软件生命周期中,软件测试往往占据了很大的比重[1].随着软件系统规模的增长,软件的逻辑结构日益复杂,软件测试所耗费的时间和生产成本也越来越多,而其中由于人工设计测试用例的成本就占据了测试总成本的 40%[2].因此,研究自动化的测试用例生成技术,节省软件测试成本,成为了时下热门的研究方向之一[3].

条件判定覆盖(condition/decision coverage,简称 C/DC)准则是白箱测试中的一种重要测试覆盖准则,由于该覆盖准则的要求比路径覆盖更为精细,且比条件覆盖效率更高,故而许多安全攸关的软件测试协议都以该覆盖准则作为判断标准.例如,飞行安全软件标准 DO-178B 就将 MC/DC(modified C/DC)列为必须满足的测试覆盖准则之一[4].因此,本文基于该准则来研究高效的测试用例生成方法.

现有的测试用例自动生成技术主要有两类:基于符号执行和基于搜索的方法.符号执行方法将程序输入定义为抽象符号,在程序执行过程中以符号运算操作代替具体的运算指令,从而收集各执行路径的符号条件约束,将测试用例生成问题转化为约束求解问题.这里的约束是以程序输入为符号变量的等式或不等式组.符号执行使用约束求解器来求解约束,但由于精度限制和非线性约束求解的复杂性,现有的约束求解器很难处理较为复杂(例如存在 SMT 不可判定的量化公式)的非线性浮点约束,这导致符号执行在遇到非线性浮点约束时覆盖效果并不理想[5].基于搜索的方法包括随机法和启发式方法.随机法通过随机地生成大量程序输入来试图满足测试覆盖准则,这种方法处理小规模的简单程序时效果较好,但随着程序规模的扩大,随机法的命中率显著下降[6].启发式搜索用一个适应度函数(fitness function)来评估搜索到的数据,并选取最佳数据作为搜索结果.这种带有导向性的搜索机制使得启发式方法的平均性能优于随机法[7].更进一步,启发式搜索又分为局部搜索和全局搜索.局部搜索(例如希尔攀登法)受制于初始输入数据的影响,容易陷入局部最优[2].全局搜索(例如模拟退火法(simulated annealing,简称 SA)、遗传算法(genetic algorithm,简称 GA))采用了特殊的控制机制,使得计算结果能更容易达到全局最优,是目前面向 C/DC 覆盖标准的最好的测试用例自动生成方法[2].已有的研究表明:在适当的配置条件下,遗传算法的搜索覆盖效果最好[2].但是,该方法仍然存在以下不足:

- 遗传算法要求在执行之前明确各个测试用例的搜索空间.当用户给出的搜索空间过大时,会极大地影响搜索效率;而当给出的搜索空间过小时,搜索空间之外的测试用例将无法覆盖.在大部分情况下,要求用户在执行测试之前给出较为精确的测试用例搜索空间是十分困难的;
- 遗传算法要求较为复杂的参数设定.在不同的测试场景下,遗传算法严格要求设定各项参数.由于遗传算法来源于生物学,所设定的参数包括编码策略、种群大小、选择策略、模拟染色体的交叉策略、交叉概率、变异概率等等.不同设定条件下的搜索性能差别很大.

针对这些问题,本文面向条件判定覆盖准则提出了一种新的线性拟合制导的测试生成方法.该方法首先将程序中的每一个判定条件规范化为一个与零值比较的数值函数,并通过插桩获得该函数在程序执行时的采样.我们利用这些采样值线性拟合数值函数,并由拟合函数估算出程序的各个判定条件与程序输入的关系,并生成测试用例.随着采样的增多,各条件判断对应的拟合函数会得到不断地细化与更新,拟合函数的定义范围也会逐步得到拓展.最终,我们沿着 C/DC 准则逐个在拟合函数上查找满足条件的目标输入区间,从而生成满足 C/DC 准则的测试用例.相对于已有的方法,该线性拟合制导的测试生成方法具有如下优点:

- 可处理复杂非线性条件约束.由于本文方法直接执行程序获得约束对应的函数采样,故而并不依赖于条件约束的形式,对于较为复杂的非线性约束,我们仍然可以通过拟合逐步逼近原始函数而使其得到正确的处理,从而找到正确覆盖的测试用例;
- 能够高效获得高覆盖率的用例.本文方法避免了复杂的符号计算,因而能够处理工业级规模的程序.相对于局部搜索和全局搜索算法,本方法直接针对程序的判定进行拟合,因而制导过程更加直接,且具有针对性.实验表明,本方法能在比现有方法更短的时间内获得更高的测试生成覆盖率;
- 不需要明确测试用例的搜索空间,搜索过程不会陷入局部最优.本文方法在生成测试用例的过程中并不需要用户明确给定搜索空间,而会自动对程序的各个拟合函数进行定义域拓展,尝试定义域外的测试用例.本文方法对各拟合函数的求解具有全局性,因此在搜索测试用例时,不会陷入局部最优;
- 参数配置简易.本文方法几乎没有需要用户精细配置的参数,用户使用简单有效.

虽然相对于传统方法,本文提出的线性拟合制导测试生成方法具有很多优势,但该方法也有其适用范围.线性拟合制导的测试生成方法仅能用于数值类型(如整型 int、长整形 long、浮点型 float、double 等)及其复合类型(如结构体、数组、容器等)的测试生成.对于存在非数值型输入(如字符串)的程序,需要先结合其他方法来对非数值型输入进行预处理,再由本文的方法来完成数值类型测试用例的生成.

我们基于 Eclipse 平台,以插件的形式设计并实现了线性拟合制导测试生成的工具原型,并在 3 个开源软件库中的 25 个真实程序上进行实验.实验结果表明:由于线性拟合制导的测试生成方法直接针对程序逻辑进行制导,在大多情况下,比经过参数优化的遗传算法制导的测试生成方法在 C/DC 准则下获得更高的覆盖率与更好的测试效率.在处理覆盖输入特别少的极值条件目标时,这一优势特别明显.

本文第 1 节将通过一个具体的例子来直观地介绍本文线性拟合制导测试生成方法的执行过程.第 2 节详细描述该方法的整体流程和具体的技术细节.第 3 节将本文提出的线性拟合制导方法与遗传算法制导的测试生成方法进行比较,在 3 个开源软件库中的 25 个真实程序上进行实验并给出实验结果.第 4 节介绍本文的相关工作.第 5 节总结全文并得出结论.

## 1 示例介绍

图 1 所示为一段以 $x$ 为输入的示例代码.其中,方框部分为插桩代码,在原始待测程序中并不存在,故而没有标定行号.

```
0:   double x; input(x);
1:   double w=x^2+1;
     write (sin(w));
2:   if sin(w)≤0 then
     write (w−10); write (cos(x)−0.2);
3:     if w>10 && cos(x)<0.2 then
4:       x+=1;
5:     else
6:       x−=1;
7:     endif
8:   endif
```

Fig.1 An example program

图 1 样例程序

在原始程序共有两个判定(第 2 行和第 3 行的两个 if)和 3 个判定条件表达式(分别为第 2 行的条件 c1:$\sin(w)\leq 0$,第 3 行的条件 c2:$w>10$ 以及第 3 行的条件 c3:$\cos(x)<0.2$),条件判定覆盖准则要求生成的测试用例覆盖程序中的每个判定,且每个条件表达式都要取到真/假值,例如生成 4 个测试用例,使条件关系表达式(c1,c2,c3)的取值为(true,true,true),(true,false,false),(false,true,true),(false,false,false),就能满足 C/DC 准则的覆盖要求,这也是基于 C/DC 准则的测试用例生成目标.当判定条件 c1,c2,c3 存在复杂非线性计算时(例如这里的三角函数操作 sin,cos 等),传统方法很难直接求解判定条件成立时的测试输入.本文的方法将各个判定条件看作黑

箱函数 $f(\vec{x})$ 与 0 的不等式(或等式),即 $f(\vec{x})\odot 0$,这里的函数自变量 $\vec{x}$ 为程序输入(向量维度为程序输入变量的个数).函数 $f(\vec{x})$ 也称为拟合目标函数;$\odot$为不等号或等号,即$\odot\in\{>,\geqslant,<,\leqslant,=,\neq\}$.因此对于图 1 的示例代码来说,各个判定条件及其对应的目标函数写成如下形式:

- c1: $\sin(x^2+1)\leqslant 0$,即 $f_1(x)=\sin(x^2+1)$;
- c2: $x^2+1-10>0$,即 $f_2(x)=x^2-9$;
- c3: $\cos(x)-0.2<0$,即 $f_3(x)=\cos(x)-0.2$.

为了生成高覆盖率的测试用例,我们希望生成不同的程序输入,使得程序中的每个判定条件表达式能取到尽可能多的真假值组合.本质上,我们只需要找到程序输入 $\vec{x}$,使各个目标函数 $f(\vec{x})$ 取到 0 值点以及 0 值点的各包围区间(例如 $f_1(x)>0$ 的区间和 $f_1(x)\leqslant 0$ 的区间).对于一般程序来说, $f(\vec{x})$ 本身可能非常复杂,本文的测试生成方法并不需要精确的求解黑箱函数 $f(\vec{x})$,而是通过插桩和制导程序的执行,获得各个判定条件所对应目标函数的采样值.这些采样值是很容易获取的,例如对于图 1 所示程序来说,我们仅需要对原始程序插桩图中方框部分的代码,即可获得对应目标函数在当前执行输入下的采样.更进一步,我们基于这些采样值进行线性拟合,从而粗略地估计出目标函数 $f(\vec{x})$ 的性质,以便更为方便地找到其对应零值点的包围区间,从而能够高效地生成高覆盖率的测试用例.

对于图 1 所示的程序样例,本文的方法将目标函数 $f_1(x),f_2(x),f_3(x)$看做未知的黑箱函数.首先,任意随机生成两个初始测试用例 $x$=7,$x$=1,并利用该测试用例执行程序.当程序执行到第 2 行和第 3 行时,我们通过执行插桩后的程序很容易获得当前执行时各拟合目标函数的采样值 $f_1(7)=\sin(w)=-0.26,f_2(7)=w-10=40,f_3(7)=\cos(x)-0.2=0.55$.使得条件 $\vec{c}(7)$ 的取值为(true,true,false),判定 $\overrightarrow{dc}(7)$ 的取值为(true,false);对于测试用例 $x$=1 来说,执行程序将获得 $f_1(1)=0.91$,而 $f_2(1)$和 $f_3(1)$由于未能被执行覆盖到对应路径而不能获得采样.从而覆盖条件 $\vec{c}(1)$ =(false,×,×),判定 $\overrightarrow{dc}(1)$ =(false,×).由这些采样数据,我们构建$f_1(x)$的拟合函数 $f_1'(x)=-0.195x+1.11$.本文的方法基于 C/DC 准则检查判定和条件的覆盖情况,对于判定 $\overrightarrow{dc}$ 来说,其第一分量 $dc_1$ 的真假值已经都被覆盖,第二分量 $dc_2$=true 还未被覆盖.从条件 $\vec{c}$ 来看,$dc_1$ 所属的条件 $c_1$ 的真假值已都被覆盖,$dc_2$ 所属条件 $c_2,c_3$ 仍未被覆盖.根据这一状况,我们的算法会以 $dc_2$=true 作为当前生成测试用例的覆盖目标.算法执行会首先使 $dc_1$=true 以保证目标判定 $dc_2$ 被执行到,这可以通过在拟合函数 $f_1'(x)$ 中搜索使得 $f_1'(x)\leqslant 0$ 的 $x$ 区间来获得(如 $x$=10).但实际上,当我们以 $x$=10 执行程序时,会得到 $f_1(10)=0.91$,从而 $dc_1(10)$仍然为 false.这时,以采样 $f_1(10)$进一步细化 $f_1'(x)$,使得拟合函数 $f_1'(x)$ 成为分段线性函数(即,由 $f_1(1),f_1(7),f_1(10)$这 3 个点计算分段线性函数的系数,使两个分段均通过 3 个采样点):

$$f_1'(x)=\begin{cases}-0.195x+1.11, & x\in[1,7)\\ 0.39x-2.99, & x\in[7,10]\end{cases}.$$

随着分段细化的不断进行, $f_1'(x)$ 会越来越接近于实际的目标黑箱函数 $f_1(x)$,从而使得我们通过搜索能够找到满足 $dc_1$=true 的输入 $x$=3.9,使得 $f_1(3,9)=-0.48,f_2(3,9)=6.21,f_3(3,9)=-0.92$, $\vec{c}$ (3.9)=(true,true,true), $\overrightarrow{dc}$ (3.9)=(true,true).利用这些采样值,我们可以进一步构建和不断细化拟合函数 $f_2'(x)$ 和 $f_3'(x)$,并搜索到覆盖 $dc_2$=true 的测试用例(这里,$x$=3.9 直接覆盖到了 $dc_2$=true).然后,算法依据 C/DC 准则设定下一个未覆盖的测试目标 $c_2$=false.同样用上述方法,通过反复迭代,最终能够生成满足 C/DC 准则的测试用例集.

## 2 线性拟合制导的测试生成方法

本节正式描述基于一般情况下,线性拟合制导的测试生成方法以及该方法的各项技术细节.本方法能够全局搜索数值类型的输入变量(如整型、浮点型),从而获得基于条件判定覆盖准则的测试用例集.对于非数值类型的输入,本方法并不直接适用.

### 2.1 整体流程

不失一般性,对于给定程序的程序输入在本文中表示为向量 $\vec{x}=(x_1,x_2,\ldots,x_n)$的形式,这里,$x_i$ 为程序第 $i$ 个输

入变量的值.程序的执行路径表示为执行语句的序列 $p=\langle s_1,s_2,\ldots,s_m\rangle$,当语句 $s_j$ 为一个条件跳转时(如分支条件、循环条件),我们称 $s_j$ 为判定语句.判定语句的判定条件无论多复杂,都可以规范化成多个条件关系表达式 $c$ 联立而成的析取范式(disjunctive normal form,简称 DNF),且析取范式中各个条件表达式为无副作用**的关系表达式 $e_1 \odot e_2$.这里,$\odot$为关系运算符$\odot \in \{>,\geqslant,<,\leqslant,=,\neq\}$.我们进一步将每个条件关系表达式 $e_1 \odot e_2$ 规范化成等价的零值比较形式 $e \odot 0$,这里,$e=e_1-e_2$.为了简便起见,我们简称条件关系表达式为条件.

**定义 1(关联路径)**. 从程序起点到当前条件关系表达式 $c$ 所在判定语句的某一条执行路径 $p$,称为条件 $c$ 的一条关联路径.

在实际程序中,每个条件关系表达式 $c$ 可能存在多条不同的关联路径,所有 $c$ 的关联路径将组成关联路径集 $P(c)$.对于其中一条关联路径,规范化条件表达式 $e \odot 0$ 的左部 $e$ 可看作由程序输入 $\vec{x}$ 经过固定的操作指令序列执行而得到的结果,因此,本文将其看做一个程序输入的函数 $f(\vec{x})$ ,称为拟合目标函数.对于实际的复杂软件,我们很难精确得到 $f(\vec{x})$ ,因而采用分段线性拟合函数 $f'(\vec{x})$ 来逼近它,并由此计算满足 C/DC 准则的测试用例.

图 2 给出了线性拟合制导测试用例生成的整体流程.整个流程以待测程序与少量随机生成的初始用例为输入,生成满足条件判定覆盖准则的测试用例.

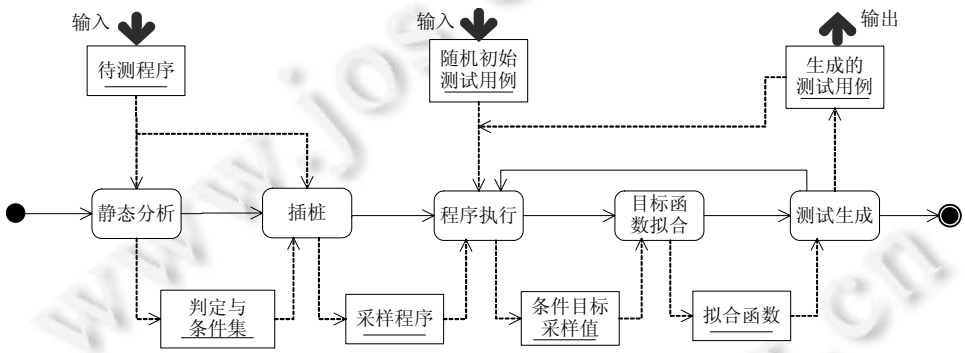



Fig.2 Main frame of linear fitting based test generation

图 2 线性拟合制导测试用例生成方法的总流程框架

从图 2 可以看出,整个框架分为 5 个步骤.前两个步骤对待测程序进行预处理,以获得各判定语句及其判定条件的集合;并通过插桩,在程序的每个判定语句前插入各拟合目标函数的采样代码,从而在执行程序时获得相应的采样值.后 3 个步骤是一个反复迭代的过程:首先,由随机生成的少量测试用例驱动程序执行,获得目标函数的采样值;然后,依据已有的采样值拟合各个判定条件的目标函数;最终,依据 C/DC 准则选择一个当前测试用例尚未覆盖到的条件表达式作为覆盖目标,从当前分段线性拟合函数中生成多个可能覆盖目标的测试用例.所生成的测试用例将会进一步驱动采样程序的执行而进入下一轮的迭代,这样既可以验证生成的测试用例是否覆盖到对应的目标,又可以进一步精化各分段线性拟合函数,使下一轮迭代生成的测试用例更为准确.这 3 个步骤的反复迭代,直到生成的测试用例集完全满足 C/DC 准则的覆盖目标.

### 2.2 待测程序的预处理

静态分析与程序插桩共同完成了对待测程序的预处理,以产生可用的采样程序.静态分析对源代码进行扫描,在遍历抽象语法树(abstract syntax tree,简称 AST)的基础上,获得当前程序的判定语句集 $S_D$ 以及各判定语句 $s \in S_D$ 的规范化判定条件 $dc$.对于存在副作用的判定条件,静态程序分析会将存在副作用的运算式从条件表达式中提出来,从而将判定条件改为无副作用的逻辑表达式.最终,判定条件 $dc$ 被静态分析模块规范化为一个条件关

** 如果程序条件中存在副作用,我们会将有副作用的运算式从条件表达式中提出来,从而将其改写成无副作用的判定条件.例如:对于判定语句 **if** $(x{+}{+})*(x{+}{+})<5$ **then** ...,我们可以将它改写成 $y=x{+}{+}$; $z=x{+}{+}$; **if** $y*z<5$ **then** ....副作用运算式的提取,会确保同原始程序在语义上一致.假如该副作用运算式会被部分程序条件逻辑短路,则提取的运算式也要加上相应的逻辑判断.

系表达式的析取范式:

$$(c_1\wedge c_2\wedge\ldots\wedge c_{n1})\vee(c_{n1+1}\wedge c_{n1+2}\wedge\ldots\wedge c_{n2})\vee\ldots\vee(c_{ni+1}\wedge c_{ni+2}\wedge\ldots\wedge c_h),$$

其中,各条件关系表达式 $c_k(1\leqslant k\leqslant h)$被规范化为零值比较形式 $e\odot 0$.这里,$e$ 为任意的算术表达式,$\odot$为关系运算符$\odot\in\{>,\geqslant,<,\leqslant,=,\neq\}$.我们可以定义判定语句的执行先后关系.

**定义 2(先执行关系)**. 对于判定语句集 $S_D$ 中的两条判定语句 $s,s_o\in S_D$,如果从程序起点到当前判定语句 $s_o$ 的每一条执行路径都经过判定语句 $s$,则称 $s,s_o$ 存在先执行关系$\preccurlyeq$,或称 $s$ 先于 $s_o$ 被执行,记为 $s\preccurlyeq s_o$.

由自反性、反对称性和传递性,我们能够证明先执行关系$\preccurlyeq$为判定语句集 $S_D$ 上的偏序关系.依据此偏序关系,在静态分析期间会构建判定语句集 $S_D$ 上的拓扑排序 $\vec{S}$ ,并输出到后续模块用于覆盖目标的选择.

在静态程序分析结果的基础上,插桩模块在待测程序各判定前插入采样代码.采样代码记录两部分信息:一是各拟合目标函数在当前执行时的采样值,这将用于函数拟合;二是各判定条件的判定结果,这将用于计算当前的执行路径,并进一步推算各个条件的关联路径,从而验证 C/DC 准则的覆盖情况.因此,采样代码也由两部分组成:第 1 部分记录当前判定中每一个规范化条件关系表达式 $e\odot 0$ 的左部值 $e$;第 2 部分记录当前整个判定条件的真假判断.根据程序执行时每个判定条件的真假判断,我们的算法即可构建当前的实际执行路径 $p$.

### 2.3 目标函数的线性拟合

在完成静态分析和插桩预处理之后,我们就开始使用随机生成的初始用例来执行插桩后的采样程序,从而获得目标函数 $f(\vec{x})$ 的采样值和各执行对于条件 $c$ 的关联路径.由第 2.1 节所述:当且仅当条件 $c$ 的多个采样值对应于 $c$ 的相同关联路径时,它们才对应于同一个目标函数 $f(\vec{x})$ .此时,才能对这多个采样值进行拟合.

#### 2.3.1 多维输入向量的拟合处理

对于多维输入向量 $\vec{x}=(x_1,x_2,\ldots,x_n)$,从其第 1 分量 $x_1$ 开始,每次仅关注其一个维度,而将其他维度上的分量设为常数进行拟合.

图 3 显示了当输入向量 $\vec{x}$ 为二维向量$(x,y)$时,线性拟合的处理过程,这里的曲面是指某目标函数 $f(\vec{x})$ 的函数图像.

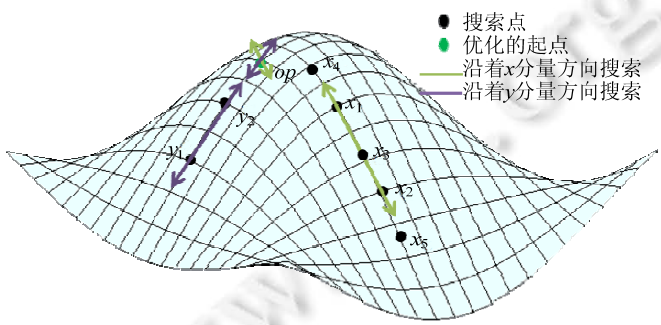



Fig.3 Fitting process for two-dimensional input vector

图 3 二维输入向量的拟合处理过程

我们首先关注其第 1 维度 $x$ 而固定其第 2 维度 $y$ 为随机常数 $y_0$,正如图 3 中绿色箭头所示:随着拟合过程的进行,$x$ 方向上会逐步采样到 $x_1,x_2,x_3,x_4,x_5$ 而构建出 $x$ 方向上的分段线性拟合函数 $f'(x,y_0)$.对于 C/DC 准则中指定的覆盖目标 $f(\vec{x})\odot 0$ 为真,可能我们在这一拟合过程中就已经找到了覆盖这一目标的程序输入向量$(x_i,y_0)$,使 $f(x_i,y_0)\odot 0$ 为真,或者我们未能找到覆盖该目标的输入向量.对于第 1 种情况,直接按照 C/DC 准则的下一个覆盖目标进入下一轮的测试生成;而对于第 2 种情况,在当前条件关系表达式 $f(\vec{x})\odot 0$ 的 $x$ 方向上一定存在 $x_{op}$ 使得分段线性拟合函数$f'(x,y_0)$的绝对值达到最小值.该向量$(x_{op}y_0)$即为固定 $y_0$ 后在 $x$ 方向上能找到的最接近于覆盖目标的用例,因此,进一步固定 $x$ 为 $x_{op}$ 而在 $y$ 方向上进行拟合,并搜索能满足覆盖目标的测试用例,如图 3

中紫色方向箭头所示.

一般地,对于多维输入向量 $\vec{x}=(x_1,x_2,\ldots,x_n)$,本方法从 $i$=1 开始,每次仅对其中一个分量 $x_i$ 做拟合,并尝试寻找满足覆盖目标 $f(\vec{x})\odot 0$ 的输入值,如果成功覆盖目标,则拟合尝试过程结束.在此过程中,除 $i$ 之外的其他各分量被固定为常数,对于后面的各分量 $x_j(j>i)$,由于该分量还未经过拟合而未能得到任何信息,故而本方法将其固定为一个随机常数.而对于每一个已经经过拟合的分量 $x_k(k<i)$,我们会在拟合 $x_k$ 时找到最接近覆盖目标的取值 $x_{op}$,使得当 $x_k=x_{op}$ 时 $|f(\vec{x})|$ 达到最小值.故在尝试拟合 $x_i$ 时,将分量 $x_k$ 固定为常量 $x_{op}$.

2.3.2 拟合函数的构建与更新

对于多维输入向量,我们在拟合过程中每次仅关注其一个维度,因此,该方法仅需要讨论在各分量维度上一维目标函数 $f(x)$的线性拟合过程.当我们以输入 $x_0,x_1$ 执行两次采样程序后,获得 $f(x)$的采样值 $f(x_0),f(x_1)$,我们即可构建拟合函数 $f'(x)=ax+b$.其中, $a=\dfrac{f(x_1)-f(x_0)}{x_1-x_0}$, $b=f(x_0)-ax_0$.

随着程序执行次数的增多,我们可能会获得目标函数 $f(x)$的更多采样值.例如:当我们进一步以 $x_2$ 来执行程序来获得采样值 $f(x_2)$后,可以将拟合函数 $f'(x)$进一步细化为两段:

$$f'(x)=\begin{cases}ax+b, & x\in[x_0,x_2)\\ cx+d, & x\in[x_2,x_1]\end{cases},$$

其中, $a=\dfrac{f(x_2)-f(x_0)}{x_2-x_0}$, $b=f(x_0)-ax_0$; $c=\dfrac{f(x_1)-f(x_2)}{x_1-x_2}$, $d=f(x_2)-cx_2$.这一更新过程如图 4 所示.

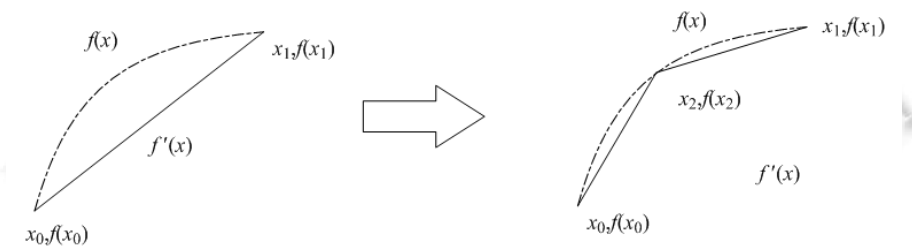



Fig.4 Updating of component linear fitting function

图 4 线性拟合函数的更新

随着我们执行采样程序次数的增多,程序中各目标函数的采样值也会越来越多,拟合函数 $f'(x)$会由于不断细化而逐步接近目标函数的取值,从而使得我们寻找的测试用例越来越准确.

2.3.3 拟合函数的自变量目标区间

拟合的最终目的在于帮助后续的测试生成,因此当获得各条件关系表达式对应的拟合函数 $f'(x)$之后,我们需要利用拟合函数根据覆盖目标 $f(x)\odot 0$ 为真或者为假获得当前的自变量目标区间.

图 5 为拟合函数 $f'(x)$的一个自变量目标区间,由于虚线所示的拟合目标函数 $f(x)$为未知的黑箱函数,我们仅能在执行过程中得到实线所示的拟合函数 $f'(x)$.由拟合函数的 0 值点(与 $x$ 轴的交点)$d,e$,我们能够获得该拟合函数针对目标 $f(x)<0$ 为真在 $x$ 分量上的自变量目标区间$(d,e)$.

**定义 3(自变量目标区间)**. 对于分段线性函数 $f'(x)$及其对应的条件关系表达式 $f(x)\odot 0$,当对于区间$(u_1,u_2)$中的每一个值 $u\in(u_1,u_2)$都满足(或都不满足)$f'(u)\odot 0$,则称区间$(u_1,u_2)$为一个针对目标 $f(x)\odot 0$ 为真(或为假)在 $x$ 分量上的自变量目标区间.

所有针对同一条件表达式目标为真(或为假)在 $x$ 分量上的自变量目标区间构成了自变量目标区间集,输入向量 $\vec{x}$ 的各分量在该条件表达式目标为真(或为假)的自变量目标区间集的并集,即为针对该覆盖目标输入向量 $\vec{x}$ 的目标区间集 $X$.

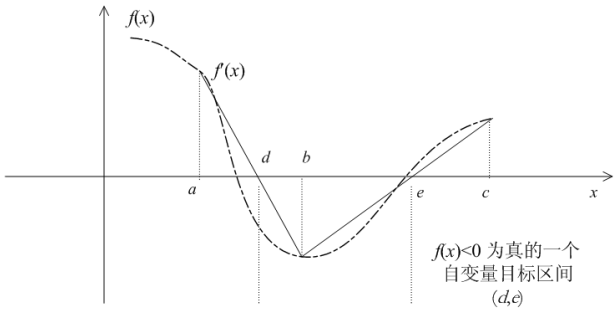



Fig.5 A target interval of the linear fitting function

图 5 拟合函数的一个自变量目标区间

2.3.4 拟合函数的细化与拓展

当拟合函数不够精确时,采用拟合函数 $f'(x)$获得的自变量目标区间存在很大误差,甚至会因此而根本找不到正确的自变量目标区间.我们通过对拟合函数的细化和拓展来解决这一问题.细化和拓展的本质是在输入的 $x$ 分量上依据一定规则进一步找一些采样点,并由这些采样点来执行程序获得新的拟合目标函数采样值,从而将拟合函数更新的更加细致(或覆盖更多的自变量范围).但如果采样点过多,一方面需要大量的程序执行消耗资源来获得采样值;另一方面也会使得分段拟合函数过于复杂,使得目标求解变得更困难.因此,细化与拓展规则对于本文的方法来说十分重要.以下介绍本文的细化和拓展规则.

**细化规则 1**. 对于分段线性拟合函数 $f'(x)$的分段$(a,f'(a)),(b,f'(b))$,当该分段直线直接存在 0 值点 $d$ 时,以 $d$ 作为细化目标,以便逼近拟合目标函数$f(x)$的真实 0 值点(如图 6 所示).

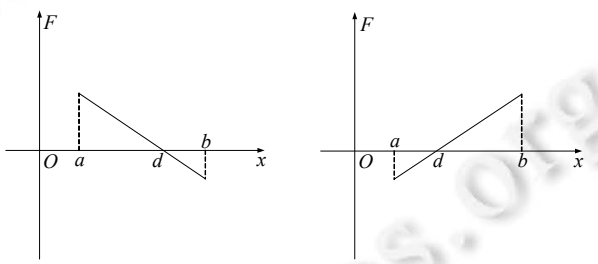



Fig.6 Generating the refinement point $d$ in the interval $(a,b)$ according to the refinement rule 1

图 6 拟合函数的分段区间$(a,b)$按细化规则 1 获得细化目标 $d$

**细化规则 2**. 对于分段线性拟合函数$f'(x)$的分段$(a,f'(a)),(b,f'(b))$,当该分段直线延长线上的 0 值点 $d$ 落在拟合函数的相邻分段$(b,c)$时,我们以 $d$ 作为细化目标.

如图 7 所示,分段$(a,b)$的延长线落在区间$(b,c)$,说明$(a,b)$的相邻分段$(b,c)$相对于当前区间$(a,b)$来说给得过于宽泛,我们可以以 $d$ 进一步细化分段区间$(b,c)$.如果延长线 0 值点 $d$ 落在相邻分段$(c,a)$,也将以 $d$ 作为细化目标.

**拓展规则 1**. 令 $l,a,b$ 为满足 $l\leqslant a<b$ 的实数.对于定义在区间$(l,b)$上分段线性拟合函数 $f'(x)$的边缘分段$(a,f'(a)),(b,f'(b))$,当该分段直线延长线上的 0 值点 $d$ 落在拟合函数的定义区间外时,我们以 $d$ 作为拓展目标.

当边缘分段延长线上的 0 值点落在拟合函数的定义区间外时,说明当前拟合函数在边缘外可能有遗漏的 0 值点,这对获得完整的自变量目标区间是不利的,因此,我们将试图以 $d$ 作为拓展目标拓展拟合函数 $f'(x)$的定义区间.

**拓展规则 2**. 令 $l,a,b$ 为满足 $l\leqslant a<b$ 的实数.对于定义在区间$(l,b)$上分段线性拟合函数 $f'(x)$的边缘分段$(a,f'(a)),(b,f'(b))$,满足拓展规则 1 的条件,即:该分段直线延长线上的 0 值点 $d$ 落在拟合函数的定义区间外,且 $d$

与拟合函数边缘 $b$ 的距离 $d-b$ 大于边缘区间宽度 $b-a$ 时,将在 $d$ 点周围宽度为 $b-a$ 的区间 $\left(d-\frac{b-a}{2},d+\frac{b-a}{2}\right)$ 上随机取一个位置点 $e$,并以 $e$ 为拓展目标.

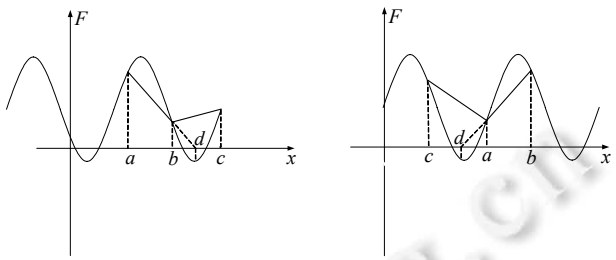



Fig.7 Generating the refinement point $d$ near the interval $(a,b)$ according to the refinement rule 2

图 7 拟合函数的分段区间$(a,b)$按细化规则 2 获得相邻分段的细化目标 $d$

图 8 显示了两个拓展规则获得拓展目标 $d$ 和 $e$ 的方式,当边缘分段直线延长线上的 0 值点 $d$ 离拟合函数边缘 $b$ 的距离过远时,分段$(b,d)$并不能清晰地反映目标函数在该区间上的细节.因此我们需要额外增加一个拓展目标 $e$,使得拓展区间更为精细.

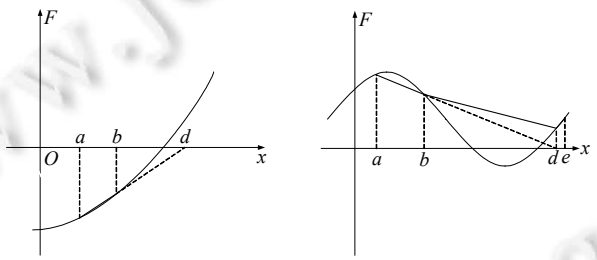



Fig.8 Extending the refinement point $d,e$ according to the extension rule 1 and 2

图 8 由拓展规则 1、拓展规则 2 获得的拓展目标 $d$ 和 $e$

### 2.4 测试生成算法

执行经过了插桩的采样程序除了能够获得采样值进行各目标函数的拟合外,还有一个重要作用在于能够验证到当前已有的测试用例集覆盖了 C/DC 准则中的哪些取值,即,哪些真假值组合已经被覆盖.对于本文方法的整体框架来说,测试生成的每一次迭代(即图 2 中的后 3 个步骤)都要求以一个 C/DC 准则中未被覆盖到的真假取值组合作为当前的测试生成目标.一般地,我们形式化定义测试生成目标为 $cc=boolval$ 的形式,其中,$cc$ 为一个规范化条件.基于 C/DC 准则,它既可以是一个规范化的判定条件 $dc$,也可以是判定条件中的一个规范化条件关系表达式 $c$.这里,$boolval\in$ {true,false}是一个布尔真假值,用于标识测试生成目标条件的取值.

由第 2.2 节可知,静态分析预处理输出判定语句的拓扑序列 $\vec{S}=(s_1,s_2,s_3,\ldots,s_m)$.对应地判定条件构成判定向量 $\overrightarrow{dc}=(dc_1,dc_2,dc_3\ldots dc_m)$,每个判定条件 $dc_i$ 规范化成如下形式的析取范式:

$$(c_1\wedge c_2\wedge\ldots\wedge c_{n1})\vee(c_{n1+1}\wedge c_{n1+2}\wedge\ldots\wedge c_{n2})\vee\ldots\vee(c_{ni+1}\wedge c_{ni+2}\wedge\ldots\wedge c_h).$$

每一个条件关系表达式目标 $c_j$=true 在对应的分段拟合函数上都存在输入向量 $\vec{x}$ 的目标区间集 $X_j$,因此,整个判定条件为真 $dc_i$=true 的目标区间集 $DX_i$ 为

$$(X_1\cap X_2\cap\ldots\cap X_{n1})\cup(X_{n1+1}\cap X_{n1+2}\cap\ldots\cap X_{n2})\cup\ldots\cup(X_{ni+1}\cap X_{ni+2}\cap\ldots\cap X_h).$$

判定条件取值为假 $dc_i$=false 的目标区间集为其补集,即:

$$\overline{(X_1\cap X_2\cap\ldots\cap X_{n1})\cup(X_{n1+1}\cap X_{n1+2}\cap\ldots\cap X_{n2})\cup\ldots\cup(X_{ni+1}\cap X_{ni+2}\cap\ldots\cap X_h)}.$$

当覆盖目标直接是判定条件的取值 $dc=boolval$ 时,对应的目标区间集即是上面给出的集合;而当测试生成

目标为判定条件 $dc$ 中的某一个规范化条件关系表达式 $c_t$ 时,由于程序设计语言条件表达式中的逻辑短路***,我们首先要保障 $c_t$ 能够被执行到.即:必须使得$(c_1\wedge c_2\wedge\ldots\wedge c_{n1})\vee\ldots\vee(c_{ns+1}\wedge c_{ns+2}\wedge\ldots\wedge c_t\wedge\ldots)\vee\ldots\vee(c_{ni+1}\wedge c_{ni+2}\wedge\ldots\wedge c_h)$中 $c_t$ 前面的每一个合取子句的取假值,且 $c_t$ 所在的合取字句中 $c_t$ 前面的每一个规范化条件关系表达式取真值.而 $c_t$ 后面的条件关系表达式和合取子句可以取任意值而忽略.因此,条件关系表达式目标 $c_t$=true 的在当前判定中目标区间集为

$$DI_t=\overline{X_1\cap X_2\cap\ldots\cap X_{n1}}\cap\overline{X_{n1+1}\cap X_{n1+2}\cap\ldots\cap X_{n2}}\cap\ldots\cap(X_{ns+1}\cap X_{ns+2}\cap\ldots\cap X_t).$$

而条件关系表达式目标 $c_t$=false 的在当前判定中的目标区间集为

$$DI_{\bar{t}}=\overline{X_1\cap X_2\cap\ldots\cap X_{n1}}\cap\overline{X_{n1+1}\cap X_{n1+2}\cap\ldots\cap X_{n2}}\cap\ldots\cap(X_{ns+1}\cap X_{ns+2}\cap\ldots\cap\overline{X_t}).$$

当我们进一步考虑关联路径的覆盖因素,设 $c_t$ 所在判定为 $dc_t$,$c_t$ 的关联路径要求判定条件向量 $\overrightarrow{dc}$ 中 $dc_t$ 前面各分量的真/假取值(具体该取真值还是假值由关联路径确定)对应的目标区间集分别为 $DX_1,DX_2,\ldots,DX_{t-1}$.故而,$c_t$=true 的总目标区间集为

$$DX_1\cap DX_2\cap\ldots\cap DX_{t-1}\cap DI_t.$$

而 $c_t$=false 的总目标区间集为

$$DX_1\cap DX_2\cap\ldots\cap DX_{t-1}\cap DI_{\bar{t}}.$$

算法 1 给出了经过待测程序预处理后,线性拟合制导测试用例生成的整体算法.在该算法中,我们用映射 $\Psi$ 来维护实际覆盖了每一个覆盖目标 $cc$=$boolval$ 的输入向量集 $\Psi(cc,boolval)$.初始的时候,由于还没有任何测试用例,$\Psi$被置空.按照 C/DC 准则,算法 1 沿着判定条件的拓扑序列依次检查各判定条件及其条件关系表达式的真假取值是否已经被覆盖到,如未被覆盖 $\Psi(cc,boolval)$==∅,则调用算法 2 定义的 *CaseGenerate* 以 $cc$=$boolval$ 为目标搜索测试用例.算法 2 以目标 $cc$=$boolval$ 生成测试用例.它对当前目标的每一条关联路径进行尝试,并对输入向量进行各维度的分量处理.通过反复尝试搜索策略,逐步细化拓展各拟合函数.在找到当前的目标区间集以后,对每一个目标区间进行随机采样,以试图覆盖算法 2 的搜索目标.最终,如经过反复迭代仍然未能覆盖目标,则返回空值.

**算法 1**. 线性拟合制导的测试用例生成算法.

输入:由静态分析预处理生成的判定条件的拓扑序列 $\overrightarrow{dc}=(dc_1,dc_2,dc_3,\ldots,dc_m)$;

输出:最终生成的测试用例集.

```
1.  ∀cc∈CC(dc⃗), Ψ(cc,true)←∅; Ψ(cc,false)←∅;      //初始化所有目标条件都还未被覆盖
2.  i←1;
3.  while i≤m
4.      for j←0; j≤cond_num(dc_i); ++j              //循环次数为 dc_i 中的条件数多 1
5.          if j==0 then
6.              cc←dc_i;                            //目标条件首先尝试当前整个判定条件
7.          else
8.              cc←c_j∈dc_i                         //然后以当前判定中各个条件关系表达式为目标
9.          end if
10.         if Ψ(cc,true)==∅ then                   //假如当前覆盖目标真值未被覆盖,则尝试生成
11.             Ψ(cc,true)={CaseGenerate(cc,true)};
12.         end if
13.         if Ψ(cc,false)==∅ then                  //假如当前覆盖目标假值未被覆盖,则尝试生成
14.             Ψ(cc,false)={CaseGenerate(cc,false)};
15.         end if
```

*** 许多程序设计语言(如 C,Java)都支持逻辑短路,即 c1∧c2∧c3 中当 c1 取假时,c2 和 c3 就不会被执行到,因此也就不会被覆盖到条件 c2 和 c3.

```
16.      end for
17.    ++i;
18.    end while
19.    return ∪Range(Ψ)                          // 返回覆盖各目标的测试用例集的并集
```

**算法 2**. 覆盖目标为 *cc*=*boolval* 的单次测试用例生成:*CaseGenerate*(*cc*,*boolval*).

```
1.   X←∅
2.   随机生成输入向量 x⃗ =(x1,x2,…,xn),用做初始随机输入
3.   for each p in P(cc)                          //对于目标的每一条关联路径
4.     i←1;
5.     while i≤n
6.       x⃗ ← x⃗ [xi/Random];                     //将输入向量 x⃗ 的第 i 个分量换成随机值
7.       X←{ x⃗ }∪X                               //将输入向量 x⃗ 加入待尝试集合 X
8.       for all x⃗ in X
9.         {sample,path}=ProgramExec( x⃗ );       //以 x⃗ 执行采样程序,获得采样值和执行路径
10.        UpdateLinearFitting(sample);          //以采样值更新各拟合函数
11.        if (cc2,boolv)=CheckCover( x⃗ ) then
12.          Ψ(cc2,boolv)←{ x⃗ }∪Ψ(cc2,boolv);   //记录 x⃗ 执行过程覆盖到的新条件
13.        end if
14.        if p in path AND eval(cc, x⃗ )==boolval then //找到满足目标的测试用例 x⃗
15.          return x⃗ ;
16.        end if
17.        X←ObjInterval(i,cc,boolval)∪X         //将当前 xi 分量上的目标区间集随机采样后加入 X
18.        X←RefExpInterval(i,p)∪X               //将关联路径上各拟合函数的细化拓展目标加入 X
19.      end for
20.      ++i;
21.    end while
22.  end for
23.  return null;                                //未能找到覆盖目标
```

## 3 实例研究

我们基于 Eclipse 平台以插件的形式设计并实现了线性拟合(linear fitting,简称 LF)制导测试生成方法的工具原型,用户在使用过程中仅需要直接将待测的 C/C++程序以 Eclipse 项目的形式导入,即可使用本文的工具自动生成满足 C/DC 规则的测试用例.由于 Eclipse 插件要求以 Java 语言实现,因此本方法的实现本身是基于 OpenJDK7 的,并通过 CDT8.1 库来操作用户待测的 C/C++程序.本文的全部实验在一台处理器为 Intel® Core™ i5-2400 3.1GHz 内存为 4GB 的工作站上运行,操作系统为 Ubuntu 13.04.

针对 C/DC 准则,本文的实验以基于遗传算法的测试用例生成技术[7,8]作为比较对象.遗传算法的测试用例生成技术是目前面向 C/DC 覆盖标准覆盖效果最好的测试用例生成方法[2].通过同遗传算法生成测试用例的比较,我们将试图回答以下几个研究问题:

- 问题 1:针对条件判定覆盖准则,线性拟合制导与遗传算法制导产生测试用例相比,覆盖效果如何?
- 问题 2:对于同样的覆盖目标,线性拟合制导与遗传算法制导相比,哪一种方法用时较短?
- 问题 3:对于传统方法较难处理的覆盖目标,例如覆盖输入特别少的目标,线性拟合与遗传算法制导的测试用例生成方法将分别有怎样的表现?

### 3.1 实验用例与参数设置

本文的实验用例来自于 3 个实际使用的开源软件库:

- Numerical Recipes(http://www.nr.com);
- Benchmark of John Burkardt(http://people.sc.fsu.edu/~jburkardt/cpp_src/cpp_src.html);
- GNU Scientific Library(http://www.gnu.org/software/gsl/).

我们保留了这些开源软件库中拥有非线性浮点运算的独立程序,并去除了带有非数值类型输入(如指针类型,字符串类型)的部分,最终得到了 25 个实用程序作为实验基准.它们的具体信息见表 1.具体给出了各个程序的名称、功能描述、代码行数(LOC)、可覆盖的判定(DE)和条件关系表达式(RE)的个数.其中,判定与条件关系的可覆盖性是由人工专门标注的.

**Table 1** Overview of the Benchmark

**表 1** 基准用例程序介绍

| 名称 | 功能描述 | LOC | DE | RE |
|---|---|---|---|---|
| julday | 根据日历计算 Julian 天数 | 34 | 4 | 4 |
| i4vec_uniform_ab_new | 获得伪随机数 I4VEC | 82 | 4 | 4 |
| ei | 获取内部指数 *Ei* | 77 | 4 | 4 |
| caldat | 根据 Julian 天数计算日期 | 48 | 6 | 6 |
| gammp | 不完备的 *gamma* 函数 | 88 | 5 | 6 |
| isValidDate | 判断一个日期是否是合理的 | 35 | 5 | 7 |
| alnorm | 计算标准正太分布的累计密度 | 91 | 5 | 7 |
| plgndr | 与球函数关联的勒让德多项式 | 48 | 5 | 7 |
| minabs | 寻找函数 *F*(*X*)=*A***abs*(*X*)+*B* 的局部极小值 | 88 | 8 | 9 |
| i4_div_rounded | 计算 I4 除法的完整的解 | 56 | 6 | 10 |
| r8_atan | 计算比率 *Y*/*X* 的反正切 | 75 | 10 | 10 |
| triangle | 对三角形进行分类 | 27 | 4 | 11 |
| sort_heap_external | 升序地对一个列表项进行排序 | 136 | 12 | 12 |
| r8_gamma | 将 *Gamma*(*X*)评估为实值 | 205 | 12 | 12 |
| r8_psi | 评估函数 *Psi*(*X*) | 200 | 11 | 12 |
| interp_cubic | 测试曲率大于 0 来提前返回极小值 | 138 | 10 | 12 |
| expint | 获取内部指数 *En* | 73 | 8 | 13 |
| index_box2_next_3d | 产生 3D 盒子的表面指数 | 75 | 6 | 14 |
| rgb_to_hue | 将(R,G,B)三原色转化成 0～1 的值 | 106 | 14 | 14 |
| approx | 计算特征值的初始近似值 | 123 | 14 | 15 |
| bessjy | 调用单元程序 *beschb* 来计算一些数值 | 165 | 16 | 17 |
| minquad | 寻找函数 *F*(*X*)=*A***X*^2+*B***X*+*C* 局部极小值 | 121 | 10 | 18 |
| gsl_ieee_set_mode | 根据给定的参数设定模式 | 138 | 19 | 19 |
| hyperg_1F1_ab_posint | 处理给定参数 *a* 和 *b* 均为正整数的情形 | 116 | 15 | 22 |
| gsl_strerror | 根据给定的错误类型输出警告信息 | 99 | 35 | 35 |

遗传算法的测试用例生成效果受参数设定的影响很大,我们在多次反复实验的基础上,选取了一组效果较好的遗传算法参数,具体见表 2.除此之外,对于每一个程序,遗传算法需要一个优化的测试用例搜索空间才能得到好的测试覆盖.我们同样经过反复调优,最终遗传算法的测试搜索输入区间以及在这些参数下遗传算法制导的对应测试覆盖率见表 3.

**Table 2** Parameter settings in genetic algorithm

**表 2** 遗传算法参数设定

| 参数 | 设定值 |
|---|---|
| 编码策略 | 16 位二进制编码 |
| 种群大小 | 50 |
| 选择策略 | 轮盘赌策略 |
| 交叉策略 | 一点交叉 |
| 交叉概率 | 0.7 |
| 变异概率 | 0.1 |

**Table 3** Search interval of genetic algorithm and the corresponding coverage

**表 3** 遗传算法的测试用例搜索区间以及对应完成搜索的覆盖率

| 名称 | GA 搜索区间 | GA 覆盖率(%) |
|---|---|---|
| julday | (−1500,1500) | 95 |
| i4vec_uniform_ab_new | (−100,100) | 100 |
| ei | (−100,100) | 50 |
| caldat | (−3E+6,3E+6) | 100 |
| gammp | (−10,10) | 66.67 |
| isValidDate | (−2000,2000) | 97.14 |
| alnorm | (−100,100) | 100 |
| plgndr | (−100,100) | 100 |
| minabs | (−100,100) | 87.78 |
| i4_div_rounded | (−100,100) | 100 |
| r8_atan | (−10,10) | 80 |
| triangle | (−100,100) | 100 |
| sort_heap_external | (−10,10) | 99.17 |
| r8_gamma | (−200,200) | 81.67 |
| r8_psi | (−4E+16,3E+15) | 33.33 |
| interp_cubic | (−30,30) | 95.83 |
| expint | (−10,10) | 83.33 |
| index_box2_next_3d | (−30,30) | 97.86 |
| rgb_to_hue | (−10,10) | 100 |
| approx | (−100,100) | 86.67 |
| bessjy | (−10,10) | 41.18 |
| minquad | (−10,10) | 67.78 |
| gsl_ieee_set_mode | (−30,30) | 100 |
| hyperg_1F1_ab_posint | (−100,100) | 95.46 |
| gsl_strerror | (−100,100) | 100 |

为了防止测试用例搜索停滞在某一个难以覆盖判定条件而影响整个测试生成的效率,我们设定判定条件目标搜索的时间阈值为 10s,如果线性拟合算法在 10s 内仍然不能搜索到覆盖当前条件目标的用例,则放弃当前目标而尝试下一个覆盖目标.

### 3.2 实验结果分析

为了评价测试用例生成算法的有效性,我们以覆盖率和测试效率作为评价指标.C/DC 准则以生成的测试用例实际覆盖到的判定和条件的数目占可覆盖的判定和条件总数的百分比作为覆盖率.当对应的算法分别生成了两个测试输入使程序执行时条件或判定取值为真和为假时,我们称对应的条件或判定被覆盖.以程序 triangle 为例,其可覆盖判定总数为 4,可覆盖条件总数为 11,如果算法覆盖了 3 个判定和 9 个条件,则其 C/DC 覆盖率为 (3+9)/(4+11)×100%=80%.除了覆盖率之外,算法完成测试用例生成的效率也是一个重要的评价因素,即,算法达到目标覆盖率所需要的时间.在我们的实验中,以两种算法达到相同覆盖率所需的时间作为评价测试效率的标准.对于每一个基准程序,两种算法会分别运行 10 次测试用例生成,并以它们的平均指标来衡量算法的有效性,以防由于一些偶然因素而引起运行结果的异常.

表 4 显示了两种算法在 25 个基准程序上生成测试用例的覆盖率和测试效率.对于每一个基准程序,我们分别用两种不同的方法进行测试用例生成,获得各自的覆盖率;然后,我们以其中较小覆盖率作为覆盖目标,再分别测定两种算法达到该覆盖率所用的时间.每一轮实验会反复运行 10 次,并以它们的平均值作为测定结果.从表 4 的数据来看:在大部分情况下,线性拟合制导测试生成方法覆盖率高于经过优化的遗传算法制导方法,仅有极少数情况下(sort_heap_external,triangle 和 interp_cubic),线性拟合制导的效果略低.这是由于该程序的条件判定正好与遗传算法的变异规则相匹配造成的.当我们对各程序按照程序规模(即,按照代码行数)加权平均来计算平均覆盖率时(表 4 最后一行),线性拟合制导方法的平均覆盖率为 94.28%,比遗传算法的 81.77%提高了约 12.5 个百分点.

当我们分析两种方法对相同覆盖率目标的测试生成时间时,我们可以看到:在很多程序(如 julday,minabs)中,线性拟合制导测试生成算法远远快于经过优化的遗传算法制导方法.而另外一些程序,两者消耗的时间较为

接近,经过加权平均,线性拟合制导方法的平均测试生成时间为 50.547s,比遗传算法的平均测试生成时间 64.514s 加快 27.6%.同时,我们比较了两种方法生成的测试用例个数,对于大部分程序,线性拟合方法以相对较少的测试用例个数获得了更好的测试覆盖.

**Table 4** Experiment results of test generation guided by linear fitting and genetic algorithm

**表 4** 线性拟合制导与遗传算法制导测试生成方法的实验结果对比

| 函数 | 覆盖率(%) | | 时间(s) | | 生成测试用例个数 | |
|---|---|---|---|---|---|---|
| | LF | GA | LF | GA | LF | GA |
| julday | 100 | 95 | 0.329 | 2.988 | 25 | 34 |
| i4vec_uniform_ab_new | 100 | 100 | 20.828 | 20.095 | 33 | 31 |
| ei | 75 | 50 | 60.121 | 70.083 | 70 | 82 |
| caldat | 100 | 100 | 0.017 | 0.02 | 26 | 35 |
| gammp | 100 | 66.67 | 64.468 | 80.822 | 80 | 92 |
| isValidDate | 100 | 97.14 | 3.714 | 7.539 | 37 | 61 |
| alnorm | 100 | 100 | 0.052 | 0.047 | 17 | 17 |
| plgndr | 100 | 100 | 11.008 | 10.025 | 28 | 25 |
| minabs | 100 | 87.78 | 1.29 | 27.243 | 19 | 66 |
| i4_div_rounded | 100 | 100 | 0.315 | 0.523 | 51 | 71 |
| r8_atan | 90 | 80 | 80.332 | 75.557 | 85 | 82 |
| triangle | 94.55 | 100 | 10.059 | 6.583 | 91 | 84 |
| sort_heap_external | 97.69 | 99.17 | 1.424 | 3.891 | 62 | 82 |
| r8_gamma | 100 | 81.67 | 31.134 | 54.34 | 85 | 91 |
| r8_psi | 95.83 | 33.33 | 32.962 | 110.265 | 53 | 103 |
| interp_cubic | 91.67 | 95.83 | 84.593 | 68.125 | 68 | 56 |
| expint | 91.67 | 83.33 | 48.292 | 79.008 | 75 | 111 |
| index_box2_next_3d | 100 | 97.86 | 33.008 | 38.629 | 72 | 75 |
| rgb_to_hue | 100 | 100 | 21.658 | 20.697 | 66 | 60 |
| approx | 86.67 | 86.67 | 22.92 | 25.485 | 75 | 79 |
| bessjy | 60 | 41.18 | 116.94 | 154.769 | 71 | 102 |
| minquad | 100 | 67.78 | 120.171 | 130.336 | 224 | 341 |
| gsl_ieee_set_mode | 100 | 100 | 179.022 | 181.275 | 319 | 366 |
| hyperg_1F1_ab_posint | 100 | 95.46 | 85.423 | 94.224 | 90 | 93 |
| gsl_strerror | 100 | 100 | 0.154 | 0.232 | 135 | 184 |
| LoC-Weighted Avg. | 94.28 | 81.77 | 50.547 | 64.514 | 72.39 | 86.73 |

在程序中,常常有一些特别难以覆盖到的判定和条件.例如在我们的实验基准程序 r8_gamma 中存在一个判定条件,要求该程序输入 $y$ 的取值在 2.23E−308~2.22E−16 之间时才可以覆盖到,这对于可以在−200~200 之间任意取值的输入 $y$ 来说是很难覆盖到的.我们把这一类在程序中很难覆盖到的条件称为极值条件.在实验中,我们手工标注了含有极值条件的 6 个基准程序,并分别检测两种测试生成方法对于极值条件的覆盖情况,实验结果如图 9 所示.

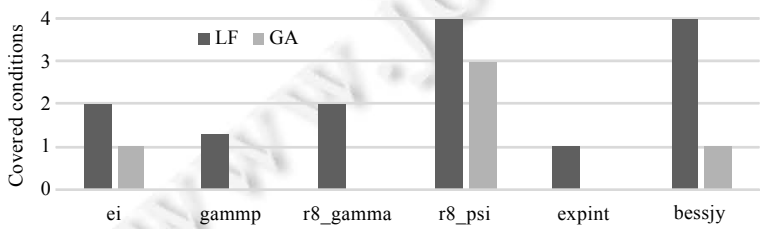



Fig.9 The covered number of extreme conditions with the approach guided by LF and GA

图 9 线性拟合制导与遗传算法制导方法的极值条件覆盖数

由图 9 的数据可知,线性拟合制导的测试用例生成算法在覆盖极值条件目标时的优势特别明显.不同于遗传算法采用类似于染色体交叉、变异的制导方式,线性拟合方法会不断逼近程序中实际条件上的拟合函数.当经过有限次迭代后,只要拟合函数足够接近极值条件的目标函数,我们就可以搜索到对应的极值条件点,从而覆盖该条件.这是传统方法很难做到的.

### 3.3 实验结论

综上所述,我们的实验结果表明:由于线性拟合制导的测试生成方法直接针对程序逻辑进行制导,因而能够获得更好的程序测试用例生成效果.对于问题 1,在大多数情况下,即使遗传算法经过了优化,线性拟合制导的测试生成方法仍然能在 C/DC 准则的标准下获得更高的覆盖率;对于问题 2,当我们以某可达的覆盖率作为覆盖目标时,线性拟合制导的测试生成方法将会以更短的测试生成时间达到覆盖目标;对于问题 3,线性拟合制导的测试生成方法在处理覆盖输入特别少的极值条件目标时,相对于遗传算法制导有特别明显的优势.

## 4 讨 论

本文提出了面向 C/DC 准则的线性拟合制导测试用例生成方法,该方法的主要优势在于,它能够处理程序中复杂的非线性条件约束.因为它将各个约束条件看作黑箱的目标函数,而采用线性拟合方法来得到一个逼近目标函数的线性拟合,因此相当于把复杂的非线性条件约束转换成了线性条件来进一步处理.因此,一个直观的测试用例生成方案是:对于程序中的普通线性条件约束,仍然采用现有的生成方法;而当遇到现有的求解器不可解的复杂非线性条件约束时,才切换到线性拟合制导测试用例生成方法来完成测试生成.本文目前并未采用这一方案,原因在于:使用求解器求解约束的测试生成方法需要一个收集约束的过程,例如符号执行过程.这一过程本身需要很大的时间开销,从而使得求解器能够得到实际的显式约束条件(即白盒约束条件),而这个收集过程在本文的方法中是可以省略的,线性拟合方法并不需要显式的白盒约束条件,从而能够极大地提高生成效率.当目标函数为线性函数时,本文方法相当于用线性参数来逼近线性的目标函数,这个过程的收敛速度很快,故而本文将测试生成过程设计成直接使用线性拟合来处理程序中所有数值约束条件的方式.

本文的测试生成方法面向 C/DC 准则尽可能生成高覆盖的测试用例,然而它并不能保证一定能够使生成的测试用例达到完全覆盖:一方面,程序中可能存在不可达的 C/DC 目标,例如当某个条件对于所有的输入恒为真时,该条件为假的 C/DC 目标就会覆盖不到;另一方面,线性拟合的采样策略会使得某些测试输入采样概率很低,当目标函数形状极为特殊时,对应的 C/DC 目标也会覆盖不到,因此在本文实验中,我们并不能使得所有程序的测试覆盖率达到 100%.软件测试本身并不是完备的方法,我们的方案能够以较少的采样次数而获得高覆盖率的效果,因此本文的方法在软件测试中是有用的.

## 5 相关工作

针对面向 C/DC 的测试数据生成问题,学者提出了多种处理方法.其中,基于启发式搜索的方法受到了很大的关注.该类方法首先对程序输入进行字符串编码将程序输入域映射到搜索域;然后,通过构建一个目标函数将测试数据生成问题转化为最优化问题[8].目前,基于启发式的搜索方法主要包括希尔攀登法(Hill climbing)、模拟退火法(simulated annealing)、基因遗传法(genetic algorithm)等.希尔攀登[9,10]是一种局部搜索方法,它从搜索空间中随机选取一个点作为初始值,然后计算该点周围其他点的适应度函数(fitness function)值,并与原始点值进行比较,从中选取最小(或最大)的对应点作为下一次搜索的起始值.希尔攀登法能够找出初始值周围中最合适的取值,但这个取值可能只是局部最优解,并非全局最优解.全局搜索方法包括模拟退火法[2,11,12]和基因遗传法[13−15],它们的搜索过程类似于希尔攀登法,但它们采用各自的机理减少陷入局部最优的可能性.其中,遗传算法通过模拟自然界生物进化的思想,通过对一组由字符串编码组成的个体进行交叉、变异等操作,不断淘汰适应度值低的个体,并生成适应度值高的个体,从而找到可以覆盖目标条件取值的程序输入.然而,遗传算法要求设定的搜索空间中必须包含满足测试覆盖准则的数据;并且它包含很多参数,参数的设定对算法的效果会产生重要影响;而且,当某些条件的有效解区间只占设定搜索空间极小部分的比例时,基因遗传算法通常不能搜索出可行的解.随机法[16−18]是一种比较简单的测试数据自动生成方法,它通过随机的生成大量的测试数据集试图去满足特定的测试覆盖标准.这种无导向的方法简单易行,比较适用于小规模的程序,但测试生成效果常常很不稳定[19,20].

符号执行技术(symbolic execution)起源于 20 世纪 70 年代[21−23],并广泛应用于程序缺陷检测、测试数据自

动生成等领域.传统的符号执行技术[21−23]通过符号推理将条件表达式转换成符号输入的不等式约束,从而将测试数据生成问题转化成约束求解问题;然后,利用约束求解器求解符号约束系统.当程序规模不断扩大时,该类方法往往存在路径爆炸问题[5].动态符号混合执行结合了符号执行和程序具体执行,扩展了符号执行的能力.典型的动态符号执行以 DART[24]和 EGT[5,25,26]为代表.DART 采用了随机的方法选择起始路径,在一定程度上避免路径爆炸问题.EGT 是另一种动态符号执行方式,它将约束系统中的一些变量替换为具体的值,然后再求解约束.这种处理方式可以在一定程度上弱化符号约束系统的求解难度,但是这种随意的赋值行为可能会屏蔽掉潜在的解区间[27].近年来,有学者在符号执行技术上提出了改进方案.例如,针对符号执行搜索策略难以覆盖程序路径的问题,并提出了各种符号执行的搜索制导方案[28−30].这些方案仍然依赖于符号执行的约束求解器来产生测试用例,因此,基于符号执行技术的测试生成方法受限于约束求解器的求解能力,理论上不存在可以求解包含任意非线性约束的约束系统的求解器[31].另外,由于约束求解器的求解精度限制,符号执行在处理含浮点变量的约束时效果不佳[32].

很多测试生成技术考虑将多种方法相结合改进生成的效果,例如在符号执行技术中引入模型检验方法[33],或者将符号执行和启发式搜索方法结合的测试数据自动生成方法[27,32,34,35].然而,这些方法面向的测试覆盖标准为语句覆盖(SC)或分支覆盖(BC),而并没有应用到条件判定覆盖上.这是由于 C/DC 准则需要考虑到判定中条件的取值,而以探索路径为指引的符号执行技术在非线性条件的处理上存在不足所致.

## 6 结束语

软件测试是工业界保障软件质量的主要手段之一,研究自动化的测试用例生成技术,对于节省测试成本具有重要意义.条件判定覆盖准则是各种安全攸关软件测试中常用的测试覆盖准则.针对这一准则,现有的自动测试生成方法存在很多不足.例如:符号执行方法很难处理较为复杂的非线性条件约束,并在处理程序的规模上受到很大限制;希尔攀登法由于在搜索过程中容易陷入局部最优而难以达到满足 C/DC 准则的高覆盖率;模拟退火法和遗传算法依赖于用户使用过程中的复杂配置,测试用例生成效果具有一定的随机性等.针对这一现状,本文提出了一种线性拟合制导的测试用例生成方法.该方法将程序中的每一个条件判定规范化为一个与零值比较的数值函数,并以插桩与执行获得该函数当前输入下的采样.通过拟合这些采样,我们能够逐步判断出程序中各个条件判定与输入的关系,并利用这些关系生成高覆盖率的测试用例.相对于传统方法,本文方法具有参数配置简易、生成过程高效等优点,并且能够处理逻辑复杂的大规模程序.我们基于 Eclipse 插件实现了线性拟合制导的测试生成工具原型,在 3 个开源软件库中的 25 个真实程序上运行的实验结果表明,本文提出的方法比目前业界测试生成效果较好的遗传算法制导方法能够获得更高的覆盖率与更好的执行效率.

**References**:

[1] Tassey G. The economic impacts of inadequate infrastructure for software testing. National Institute of Standards and Technology, RTI Project, 2002,7007(011).

[2] Xiao M, El-Attar M, Reformat M, Miller J. Empirical evaluation of optimization algorithms when used in goal-oriented automated test data generation techniques. Empirical Software Engineering, 2007,12(2):183−239. [doi: 10.1007/s10664-006-9026-0]

[3] Bertolino A. Software testing research: Achievements, challenges, dreams. In: Proc. of the 2007 Future of Software Engineering. IEEE Computer Society, 2007. 85−103. [doi: 10.1109/FOSE.2007.25]

[4] RTCA. DO-178B software considerations in airborne systems and equipment certification, radio technical commission for aeronautics (RTCA). European Organization for Civil Aviation Electronics (EUROCAE). 1992.

[5] Cadar C, Sen K. Symbolic execution for software testing: Three decades later. Communications of the ACM, 2013,56(2):82−90. [doi: 10.1145/2408776.2408795]

[6] Majumdar R, Sen K. Hybrid concolic testing. In: Proc. of the 29th Int'l Conf. on Software Engineering. Washington: IEEE Computer Society, 2007. 416−426. [doi: 10.1109/ICSE.2007.41]

[7] Bueno PMS, Jino M. Automatic test data generation for program paths using genetic algorithms. Int'l Journal of Software Engineering and Knowledge Engineering, 2002,12(6):691−709. [doi: 10.1142/S0218194002001074]

[8] Michael CC, McGraw G, Schatz M. Generating software test data by evolution. IEEE Trans. on Software Engineering, 2001,27(12): 1085−1110. [doi: 10.1109/32.988709]

[9] Korel B. Automated software test data generation. IEEE Trans. on Software Engineering, 1990,16(8):870−879. [doi: 10.1109/32.57624]

[10] Korel B. Dynamic method for software test data generation. Software Testing, Verification and Reliability, 1992,2(4):203−213. [doi: 10.1002/stvr.4370020405]

[11] 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]

[12] Ghani K, Clark JA. Automatic test data generation for multiple condition and MCDC coverage. In: Proc. of the 4th Int'l Conf. on Software Engineering Advances (ICSEA 2009). IEEE, 2009. 152−157. [doi: 10.1109/ICSEA.2009.31]

[13] Xanthakis S, Ellis C, Skourlas C, Le Gall A, Katsikas S, Karapoulios K. Application of genetic algorithms to software testing. In: Proc. of the 5th Int'l Conf. on Software Engineering and its Applications. 1992. 625−636.

[14] McMinn P. IGUANA: Input generation using automated novel algorithms. A plug and play research tool. Technical Report, CS-07-14, Department of Computer Science, University of Sheffield, 2007.

[15] Janikow CZ, Michalewicz Z. An experimental comparison of binary and floating point representations in genetic algorithms. In: Proc. of the ICGA. 1991. 31−36.

[16] Linger RC. Cleanroom software engineering for zero-defect software. In: Proc. of the 15th Int'l Conf. on Software Engineering. IEEE Computer Society Press, 1993. 2−13. [doi: 10.1109/ICSE.1993.346060]

[17] Ramamoorthy CV, Ho S-BF, Chen W. On the automated generation of program test data. IEEE Trans. on Software Engineering, 1976,3(4):293−300. [doi: 10.1109/TSE.1976.233835]

[18] Thevenod-Fosse P, Waeselynck H. STATEMATE applied to statistical software testing. In: Proc. of the ACM SIGSOFT Software Engineering Notes. ACM Press, 1993. 99−109. [doi: 10.1145/154183.154262]

[19] Korel B. Automated test data generation for programs with procedures. In: Zeil SJ, Tracz W, eds. Proc. of the ACM SIGSOFT Software Engineering Notes. ACM Press, 1996. 209−215. [doi: 10.1145/229000.226319]

[20] McMinn P. Search-Based software testing: Past, present and future. In: Proc. of the 4th IEEE Int'l Conf. on Software Testing, Verification and Validation Workshops (ICSTW). IEEE, 2011. 153−163. [doi: 10.1109/ICSTW.2011.100]

[21] Boyer RS, Elspas B, Levitt KN. SELECT—A formal system for testing and debugging programs by symbolic execution. ACM SigPlan Notices, 1975,10(6):234−245. [doi: 10.1145/390016.808445]

[22] Howden WE. Symbolic testing and the DISSECT symbolic evaluation system. IEEE Trans. on Software Engineering, 1977,3(4): 266−278. [doi: 10.1109/TSE.1977.231144]

[23] King JC. Symbolic execution and program testing. Communications of the ACM, 1976,19(7):385−394. [doi: 10.1145/360248.360252]

[24] Godefroid P, Klarlund N, Sen K. DART: Directed automated random testing. In: Proc. of the ACM Sigplan Notices. ACM Press, 2005. 213−223. [doi: 10.1145/1065010.1065036]

[25] Cadar C, Ganesh V, Pawlowski PM, Dill DL, Engler DR. EXE: Automatically generating inputs of death. ACM Trans. on Information and System Security (TISSEC), 2008,12(2):10. [doi: 10.1145/1455518.1455522]

[26] Cadar C, Dunbar D, Engler DR. KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs. In: Proc. of the OSDI. 2008. 209−224.

[27] Dinges P, Agha G. Solving complex path conditions through heuristic search on induced polytopes. In: Proc. of the 22nd ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. ACM Press, 2014. 425−436. [doi: 10.1145/2635868.2635889]

[28] Li Y, Su Z, Wang L, Li X. Steering symbolic execution to less traveled paths. ACM SigPlan Notices, 2013,48(10):19−32. [doi: 10.1145/2544173.2509553]

[29] Marinescu PD, Cadar C. KATCH: High-Coverage testing of software patches. In: Proc. of the 9th Joint Meeting on Foundations of Software Engineering. ACM Press, 2013. 235−245. [doi: 10.1145/2491411.2491438]

[30] Su T, Pu G, Fang B, He J, Yan J, Jiang S, Zhao J. Automated coverage-driven test data generation using dynamic symbolic execution. In: Proc. of the 8th Int'l Conf. on Software Security and Reliability (SERE). 2014. 98−107. [doi: 10.1109/SERE.2014.23]

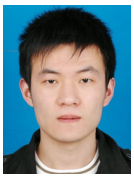
[31] Davis M. Hilbert's Tenth Problem is Unsolvable. American Mathematical Monthly, 1973. 233−269. [doi: 10.2307/2318447]

[32] Lakhotia K, Tillmann N, Harman M, De Halleux J. Flopsy-Search-Based floating point constraint solving for symbolic execution. In: Petrenko A, Simão A, Maldonado JC, eds. Proc. of the Testing Software and Systems. Springer-Verlag, 2010. 142−157. [doi: 10.1007/978-3-642-16573-3_11]

[33] Su T, Fu Z, Pu G, He J, Su Z. Combining symbolic execution and model checking for data flow testing. In: Proc. of the 37th Int'l Conf. on Software Engineering. ICSE, 2015. [doi: 10.1109/ICSE.2015.81]

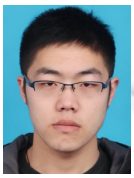
[34] Inkumsah K, Xie T. Evacon: A framework for integrating evolutionary and concolic testing for object-oriented programs. In: Proc. of the 22nd IEEE/ACM Int'l Conf. on Automated Software Engineering. ACM Press, 2007. 425−428. [doi: 10.1145/1321631.1321700]

[35] Tillmann N, De Halleux J. Pex—White box test generation for net. In: Proc. of the Tests and Proofs. Springer-Verlag, 2008. 134−153. [doi: 10.1007/978-3-540-79124-9_10]



**汤恩义**(1982－),男,江苏苏州人,博士,助理研究员,主要研究领域为软件工程,新型软件测试方法,程序分析方法.

**欧建生**(1989－),男,硕士,主要研究领域为软件工程,自动化单元测试.



**周岩**(1991－),男,主要研究领域为软件测试.

**陈鑫**(1975－),男,博士,讲师,主要研究领域为软件工程,软件测试和验证技术.