文章编号:1007-5321(2017)05-0067-08 **DOI**:10.13190/j.jbupt.2016-284

基于缺陷检测难度的测试用例检错能力模型

谭立力[1], 王雅文[1], 邢 颖[2], 王 前[1]
(1.北京邮电大学 网络与交换国家重点实验室, 北京 100876; 2.北京邮电大学 自动化学院, 北京 100876)

摘要: 在经典的变异评分计算过程中,因为不考虑被播种软件缺陷的检测难度而使得变异评分的可信性受到质疑. 因此提出一种基于缺陷检测难度评价测试用例集合的方法. 以 logistic 回归为基础,利用经验回归方程建立缺陷的识别概率与缺陷检测难度之间的数量关系. 借助关系曲线下的面积,变异评分被重新定义. 新定义的变异评分不但不受缺陷样本的检测难度影响,而且规避了因等价变异体的出现而使得经典变异评分的计算不准确的问题.

关 键 词: 软件可测试性; 变异测试; 随机测试用例生成; logistic 回归
中图分类号: TP311.52 **文献标志码**: A

Modeling Test Case Fault-detecting Capacity Based on Fault Detection Difficulty Level

TAN Li-li[1], WANG Ya-wen[1], XING Ying[2], WAN Qian[1]
(1. State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China;
2. Automation School, Beijing University of Posts and Telecommunications, Beijing 100876, China)

Abstract: The credibility of mutation score is questioned because it is computed without regard to fault detection difficulty level. A fault-detecting capacity evaluation model about test suit considering fault detection difficulty level is suggested. Based on logistic regression, the quantitative relation between the fault recognition probability and fault detection difficulty level is expressed. With the area under this relation curve, the mutation score is redefined. The new defined mutation score is not affected by fault detection difficulty level and avoid inaccuracy of classical mutation score because of the appearance of equivalent mutants.

Key words: software testability; mutation testing; random test case generation; logistic model

如何度量测试用例集合的检错能力是软件测试的核心问题之一. 变异测试技术[1-3]被研究者提出,用于衡量测试用例集合的检错能力. 变异测试使用变异算子对被测试语句产生语法变异,将生成的变异缺陷作为现实世界软件缺陷的代表[4-5],利用变异评分评价测试用例集合的充分性[6-7]. 此外变异测试的研究者也开发出一些变异测试工具[8-9].

AJ Offutt 等[10]指出测试用例集合的变异评分与被播种的软件缺陷的检测难度密切相关. 因此在检测难度未知的缺陷集上计算出的变异评分受到质疑.

在提出的缺陷失效模型中,将软件缺陷被识别

收稿日期: 2016-12-02
基金项目: 国家自然科学基金项目(61202080); 广西云计算与大数据协同创新中心、广西高校云计算与复杂系统重点实验室资助(YD16508)
作者简介: 谭立力(1976—), 男, 博士生, E-mail: tanlili@bupt.edu.cn; 王雅文(1983—), 女, 副教授.

出的概率看成以下3个概率的乘积:执行到一个软件缺陷的概率(执行概率),产生一个不正确的数据状态的概率(传染概率),把这个不正确的数据状态传播给输出的概率(传播概率)[11]. Voas等[12]还提出利用动态分析方法,在被测试程序的输入域内以均匀随机的方式产生测试用例,利用这些随机测试用例的执行结果来估计缺陷的检测难度.

为了度量基于代码覆盖的测试用例集合的检错能力, Chen[13]等研究者提出了表达式的缺陷暴露率. 表达式的缺陷暴露率被定义为在程序的输入域内,以随机的方式产生并且能够执行到该表达式的所有测试用例里,能够识别出此表达式内存在的缺陷的测试用例的比例.

虽然变异评分和各种缺陷检测难度被分别研究过,但至今研究者还没有给出一个函数,使之刻画缺陷的识别概率随缺陷检测难度的变化所呈现出的规律.

于是笔者提出了一种评价测试用例集合的充分性的方法. 此法借助回归模型建立缺陷的识别概率与缺陷的检测难度之间的函数关系. 首先基于路径输入域生成均匀分布的随机测试用例并将其执行,以此计算缺陷检测难度. 然后执行待评估的测试用集合,记录缺陷检测结果. 最后以缺陷的检测难度和检测结果做数据样本,借助logistic回归建立缺陷的识别概率与缺陷检测难度之间的数量关系. 这种数量关系以曲线形式表达,反映了测试用例集合的检错能力. 借助关系曲线之下的面积,每一个表达式被检测的充分性被定义. 此充分性被称为表达式的变异评分. 基于表达式的变异评分,被测试函数的变异评分被重新定义. 它是被测试程序中所有表达式的变异评分的平均值.

1 建模测试用例集合的检错能力的基本思路

测试用例集合的检错能力模型的建模过程由两大部分组成,第1部分是缺陷检测数据的获取,第2部分是建立与拟合检错能力模型.

1.1 缺陷检测数据的获取

提出了一种软件缺陷检测数据的获取方案. 方案涉及2个方面:第1方面是缺陷的检测难度的计算,第2方面是从缺陷的检测结果中提取有效数据.

1) 缺陷的检测难度的计算

缺陷检测难度的计算过程如下:首先以不可重复的随机抽样方式在被测试函数的控制流程图中选中一部分表达式 $e_i, i=1,2,\cdots,I$. 将这些表达式构成的样本集合记为 S. 接着以集合 S 中的每一个表达式 e_i 为目标,在控制流程图上以可重复的随机方式进行 h 条完整路径的选择,并在每一条选中的路径内以均匀随机的方式生成1个测试用例. 此测试用例称为以表达式 e_i 为目标的均匀随机测试用例. 在集合 S 的每一个表达式 e_i 上,以随机的方式播种 n_i 个缺陷 $m_{ij}, i=1,2,\cdots,I, j=1,2,\cdots,n_i$. 最后把以表达式 e_i 为目标的所有均匀随机测试用例中能够识别缺陷 m_{ij} 的测试用例的比例 x_{ij} 称为软件缺陷 m_{ij} 的暴露概率. 缺陷的暴露概率越大表明缺陷的检测难度越低.

在上述过程中,生成均匀分布的测试用例的具体方法如下. 首先将路径约束分解为多个不含否定的简单合取范式的析取结构. 然后随机选择一个合取范式作为路径约束条件的代表. 当路径代表约束只包含一次算数表达式时,路径代表约束表现为一个凸多面体. 此时调用R语言开源软件包hitandrun. 开源软件hitandrun利用马尔可夫蒙特卡洛方法在线性约束构成的凸多面体内实现了均匀随机抽样[14-15]. 随后将均匀随机抽样的数据封装成测试用例. 在利用开源软件进行均匀随机抽样之前,要在路径代表约束中添加测试用例的取值上界与下界,以防止生成的测试用例在执行过程中发生溢出.

2) 缺陷检测结果的提取

将待评估的测试用例集合 T 中的测试用例按照生成的先后顺序在软件缺陷 m_{ij} 上执行. 如果缺陷 m_{ij} 不能被在其上执行的前 $k-1$ 个测试用例识别的条件下,依然不被第 k 个测试用例识别,那么记检测结果 $y_{ijk}=1$,否则记 $y_{ijk}=0$. 称此种缺陷检测结果被称为基于条件的缺陷检测结果. 当缺陷 m_{ij} 被执行在其上的第 L 个测试用例首次检测到时,利用缺陷 m_{ij} 获得的基于条件的缺陷检测结果总数 h_{ij} 等于 L,否则缺陷 m_{ij} 不能被测试用例集合检测到,h_{ij} 等于在缺陷 m_{ij} 上执行过的测试用例个数 l_i. 此方法使得 S 上一共产生

$$|S| = \sum_{i=1}^{I}\sum_{j=1}^{n_i} h_{ij} \tag{1}$$

个基于条件的缺陷检测结果.

将具有暴露概率为 x_{ij} 的缺陷 m_{ij} 不被在其上执行的前 $k-1$ 个测试用例识别的条件下，不能被第 k 个测试用例识别的概率简称为缺陷 m_{ij} 的第 k 个不能被识别的条件概率，记为 $q_{ij}(x_{ij},k)$.

1.2　测试用例集合的检错能力模型的构建与拟合策略

使用条件概率对测试用例集合的检错能力进行建模的原理如下.

暴露概率为 x_{ij} 的缺陷 m_{ij} 不能被识别的概率

$$q_{ij}(x_{ij})=\prod_{k=1}^{l_i}q_{ij}(x_{ij},k) \tag{2}$$

其中 $q_{ij}(x_{ij},1)$ 被定义为缺陷 m_{ij} 被第 1 个测试用例执行时不能被识别的概率. 为了便于分析，假定缺陷不能被识别的条件概率只由缺陷的检测难度 x 和执行在该缺陷上的测试用例序号 k 决定. 假定表达式 e_i 被 l_i 个测试用例执行过，则可以使用函数曲线 $q(x)$ 下的面积与单位 1 的差距

$$p_i=1-\int_{x=0}^{1}q(x)\,\mathrm{d}x=$$
$$1-\int_{x=0}^{1}\prod_{k=1}^{l_i}q_{ij}(x,k)\,\mathrm{d}x \tag{3}$$

代表表达式 e_i 被测试的充分性，即表达式 e_i 的变异评分. 进而可以使用集合 S 中的所有表达式的变异评分的平均值

$$p=\frac{1}{I}\sum_{i=1}^{I}p_i \tag{4}$$

重新定义被测程序的变异评分.

1）logisitc 回归模型　当软件缺陷 m_{ij} 被测试用例序列执行时，测试用例序列不能识别出该缺陷的概率可以表示为逐个测试用例不能识别出该缺陷的条件概率 q_{ijk} 的乘积. 因为条件概率 q_{ijk} 位于 0 与 1 之间，所以最初想法是采用 logistic 回归模型[16]建立条件概率 q_{ijk} 与缺陷暴露概率 x_{ij} 和缺陷上的测试用例执行序号 k 之间的回归关系

$$\mathrm{logit}(q_{ijk})=\log\frac{q_{ijk}}{1-q_{ijk}}=\eta_{rs}(x_{ij},k) \tag{5}$$

即

$$q_{ijk}=\mathrm{logit}^{-1}[\eta_{rs}(x_{ij},k)] \tag{6}$$

在 logisitc 回归模型理论中，η_{rs} 被称为线性预测器，是以数字 β_0 和向量 $\boldsymbol{\beta}$ 为系数，以 x_{ij} 和 k 为自变量的多项式

$$\eta_{rs}(x_{ij},k)=\beta_0+\sum_{u=1}^{r}\beta_u x_{ij}^u+\sum_{v=1}^{s}\beta_{r+v}k^v+$$
$$\sum_{v=1}^{s}\sum_{u=1}^{r}\beta_{s+vr+u}x_{ij}^u k^v \tag{7}$$

其中：r 与 s 分别代表了 x_{ij} 和 k 的最高次数，不同的 r 与 s 对应了不同的 logistic 模型 c_{rs}. 例如

$$\eta_{12}(x_{ij},k)=\beta_0+\beta_1 x_{ij}+\beta_2 k+\beta_3 k^2+\beta_4 x_{ij}k+\beta_5 x_{ij}k^2.$$

1）预变换　但上述 logistic 回归模型中存在缺点. η_{rs} 是关于 x_{ij} 和 k 的普通多项式，不能保证 x_{ij} 趋近 0 时 q_{ijk} 一定趋近 1，也不能保证 x_{ij} 趋近 1 时，q_{ijk} 一定趋近 0. 于是我们提出如下数据变换方法来解决这个问题：令

$$z_{ij}=\log[(1-x_{ij})/x_{ij}] \tag{8}$$

再以 z_{ij} 代替 x_{ij} 进入线性预报器 η_{rs}，构造 logistic 回归

$$\mathrm{logit}(q_{ijk})=\eta_{rs}(z_{ij},k) \tag{9}$$

其中

$$\eta_{rs}(z_{ij},k)=$$
$$\beta_0+\sum_{u=1}^{r}\beta_u z_{ij}^u+\sum_{v=1}^{s}\beta_{r+v}k^v+$$
$$\sum_{v=1}^{s}\sum_{u=1}^{r}\beta_{s+vr+u}z_{ij}^u k^v \tag{10}$$

并要求 r 为奇数. 这时如果将 q_{ijk} 视为以 z_{ij} 为自变量的表达式，不论 z_{ij} 趋近于正无穷或者负无穷，q_{ijk} 的取值都由 z_{ij}^r 的系数决定. 当此系数大于零时，

$$\lim_{x_{ij}\to 0^+}q_{ijk}=\lim_{z_{ij}\to+\infty}q_{ijk}=1$$
$$\lim_{x_{ij}\to 1^-}q_{ijk}=\lim_{z_{ij}\to-\infty}q_{ijk}=0 \tag{11}$$

当此系数小于零时，

$$\lim_{x_{ij}\to 0^+}q_{ijk}=\lim_{z_{ij}\to+}q_{ijk}=0$$
$$\lim_{x_{ij}\to 1^-}q_{ijk}=\lim_{z_{ij}\to-\infty}q_{ijk}=1 \tag{12}$$

根据物理意义，回归样本的缺陷检测数据能够呈现出以下趋势：缺陷 m_{ij} 不能被第 k 个测试用例识别的条件概率 q_{ijk} 随缺陷暴露概率 x_{ij} 的增加而减小. 这种现象使得 z_{ij}^r 的系数必然大于零，以致式(9)满足 x_{ij} 趋近 0 时，q_{ijk} 趋近 1，并且有 x_{ij} 趋近 1 时，q_{ijk} 趋近 0.

当式(9)中的系数 $\beta_0,\boldsymbol{\beta}$ 的估计值 $\hat{\beta}_0,\hat{\boldsymbol{\beta}}$ 确定之后，可以利用式(8)的逆变换，将 q_{ijk} 重新表达成 x_{ij} 与 k 的函数.

2）惩罚极大似然估计

经过数据预处理之后的 logistic 回归模型以

$$l(\beta_0,\boldsymbol{\beta})=$$
$$\sum_{i=1}^{I}\sum_{j=1}^{n_i}\sum_{k=1}^{h_i}(y_{ijk}\eta_{rs}(z_{ij},k)-$$
$$\log\{1+\exp[\eta_{rs}(z_{ij},k)]\}) \tag{13}$$

表达缺陷检测结果出现的联合概率的对数似然. 将对数似然极大化可以获得 logistic 回归模型的参数 β_0,$\boldsymbol{\beta}$ 的估计值 $\hat{\beta}_0$,$\hat{\boldsymbol{\beta}}$,从而得到 logisitc 回归模型 $c_{rs}(z,k)$ 的经验回归方程

$$\hat{q}_{rs}(z,k)=\text{logit}^{-1}[\hat{\eta}_{rs}(z,k)] \tag{14}$$

虽然对基于条件的缺陷检测结果的联合概率的极大化可以实现 logistic 回归模型的参数估计,但是 logistic 回归常常使得拟合出的经验回归方程的震荡幅度大、震荡频率高于真实情况,即引发过拟合现象.

为了防止过拟合,希望采用带有惩罚极大似然估计功能的计算软件求解 logistic 模型的系数参数. 使用惩罚极大似然估计方法对联合概率 $l(\beta_0,\boldsymbol{\beta})$ 极大化等价于极小化问题

$$\underset{\boldsymbol{\beta}}{\text{argmin}}-l(\beta_0,\boldsymbol{\beta})+\frac{1}{2}\lambda\|\boldsymbol{\beta}\|_2 \tag{15}$$

其中 λ 可以使用交叉验证方法加以选择.

3) 交叉验证

交叉验证[16]是通过测量经验回归方程的预测误差达到评价模型准确性的手段. 交叉验证不但有利于确定模型中的未知参数,还可以比较不同的回归模型的相对优劣.

在利用惩罚 logistic 回归确定未知参数的过程中,通过确定最优化的惩罚参数 λ_{opt} 完成交叉验证过程. 最常用的交叉验证是 Ψ 折交叉验证. Ψ 折交叉验证首先将基于条件的缺陷检测结果分成 Ψ 份,然后依次拿出第 $\varphi\in(1,2,\cdots,\Psi)$ 份基于条件的检测结果作为测试集,其余 $\varphi-1$ 份基于条件的缺陷检测结果作为训练集. 在 λ 给定条件下,为了在每一个测试集上检验回归效果,一共得到 Ψ 个经验回归方程 $\hat{f}_{-1},\hat{f}_{-2},\cdots,\hat{f}_{-\Psi}$. Ψ 折交叉验证的得分

$$v(\lambda)=\frac{1}{\Psi}\sum_{\varphi=1}^{\Psi}v_{\varphi}(\lambda)= \tag{16}$$

$$\frac{1}{|S|}\sum_{i=1}^{I}\sum_{j=1}^{n_i}\sum_{k=1}^{h_i}l[y_{ijk},\hat{f}_{-\varphi(ijk)}(x_{ij},k,\lambda)]$$

其中 $V_{\varphi}(\lambda)$ 表示在给定 λ 的条件下回归模型在测试集 φ 上的预测的平均损失. $|S|$ 代表按照式(1)定义的基于条件的缺陷检测结果的数量,$\hat{f}_{-\varphi(ijk)}$ 代表除去 y_{ijk} 所在的测试集 $\varphi(ijk)$ 后,使用其余 $\Psi-1$ 个训练集拟合出的经验回归方程. 测试集上暴露概率为 x_{ij} 的缺陷被第 k 个测试用例执行时缺陷不能被识别的条件概率的预测值使用 $\hat{f}_{-\varphi(ijk)}(x_{ij},k,\lambda)$ 表示. 对于 logistic 回归而言,每一个样本的预测值的损失[17]为

$$\begin{aligned}&l[y_{ijk},\hat{f}_{-\varphi(ijk)}(x_{ijk},k,\lambda)]=\\&-2\log\hat{f}^{y_{ijk}}_{-\varphi(ijk)}(x_{ijk},k,\lambda)\\&-2\log[1-\hat{f}_{-\varphi(ijk)}(x_{ijk},k,\lambda)]^{1-y_{ijk}}\end{aligned} \tag{17}$$

当 λ 的可选范围属于有限集合$\{\lambda_1,\lambda_2,\cdots,\lambda_F\}$时,$v(\lambda_1),v(\lambda_2),\cdots,v(\lambda_F)$ 中最小者对应的 λ 作为惩罚 logistic 回归中最优化的惩罚系数 λ_{opt}.

对于多个模型而言,如果每一个模型都采用惩罚性极大似然估计方法确定未知参数,则可以利用交叉验证得分比较不同模型的准确性. 因为交叉验证得分代表了模型的预测效果的损失程度,所以在候选模型中以交叉验证得分最小者为佳.

4) 拟合优度检验

可以通过把实际获得的观测数据与利用经验回归方程生成的仿真数据进行对比,完成欠拟合程度的评估[18]. 其根本思想如下:依据经验回归方程构建模型因变量(或因变量的某一个函数)出现的主要区间,如果观测样本(或观测样本的上述函数)不出现在这个主要区间内,则认为经验回归方程不能代表真实的潜在规律,从而拒绝经验回归方程,否则接受经验回归方程.

根据上述思想,设计了一个检验经验回归方程的假设检验方案. 首先假定候选模型的经验回归方程恰当地反映了潜在的客观规律,以此作为原假设. 对于已经观测到的暴露概率为 x_{ij} 的缺陷 m_{ij},基于式(8)和式(14)以及这个被假设成恰当的经验回归方程 $\hat{q}_{rs}(x,k)$ 计算出缺陷 m_{ij} 不能被其上执行的前 $k-1$ 个测试用例识别的条件下,不能够被第 k 个测试用例识别的条件概率

$$\hat{q}_{ijk}(x_{ij},k)=\text{logit}^{-1}[\hat{\eta}_{rs}(z_{ij},k)]$$

其中 $z_{ij}=\log[(1-x_{ij})/x_{ij}]$.

被观测到缺陷 m_{ij} 的基于条件的识别结果 y_{ijk},可以看成是以 $\hat{q}_{ijk}$ 为事件发生概率的二元随机变量 y^*_{ijk} 的一个典型实现,$i=1,2,\cdots,I,j=1,2,\cdots,n_i,k=1,2,\cdots,h_i$. 以致所有被观测到的 y_{ijk} 的函数 T^* 可以看成是相应的以 $\hat{q}_{ijk}$ 为事件发生概率的取值为0或1的随机变量 y^*_{ijk} 的函数 T^* 的一个实现. 当以所有的随机变量 y^*_{ijk} 构造拟合优度检验统计量

$$T^*=\sum_{i=1}^{I}\sum_{j=1}^{n_i}\sum_{k=1}^{h_i}\frac{(y^*_{ijk}-\hat{q}_{ijk})^2}{\hat{q}_{ijk}(1-\hat{q}_{ijk})} \tag{18}$$

时，所有被观测到的基于条件的缺陷识别结果 y_{111}，…，$y_{In_Ih_I}$的函数

$$T_{\text{obs}} = \sum_{i=1}^{I}\sum_{j=1}^{n_i}\sum_{k=1}^{h_i}\frac{(y_{ijk}-\hat{q}_{ijk})^2}{\hat{q}_{ijk}(1-\hat{q}_{ijk})} \tag{19}$$

构成了上述拟合优度检验统计量 T^* 的一个观测值. 在原假设成立的情况下，T_{obs}会以大概率成为 T^* 的典型代表. 当它不能成为 T^* 的典型代表时，应拒绝原假设，丢弃对应候选模型.

2　测试用例集合的检错能力的自动化建模工具介绍

以自动化单元测试系统 CTS[19] 作为基础，建模工具的工作过程分成 6 个阶段：第 1 阶段获得缺陷检测数据；第 2 阶段对缺陷检测数据进行预变换；第 3 阶段建立多个 logistic 候选回归模型并进行参数估计；第 4 阶段进行后变换得到以缺陷的检测难度为自变量的候选模型的经验回归方程；第 5 阶段对候选模型的经验回归方程进行拟合优度检验；第 6 阶段选择最优经验回归方程，计算新型变异评分.

在第 1 阶段，当被测试的 C 语言程序源文件和测试用例集合被输入到自动化建模工具之后，自动化建模工具在控制流程图上以不可重复的随机抽样方式选定 50 个表达式，构成样本集合 S. 当被测试程序的表达式总数小于 50 个时，选中全部表达式构成样本集合 S.

然后以样本集合 S 中的每一个表达式 e_i 为目标，按照随机方式在控制流程图上选定 5 条完整路径，调用 R 语言软件 hitandrun，在每一条路径的输入域内以均匀随机的方式生成 1 个测试用例.

接着自动化建模工具调用 CTS 中的缺陷注入功能，利用 12 类精简的 C 语言变异算子[20] 在样本集合 S 的每一个表达式上实施一阶变异，产生对应的缺陷文件 m_{ij}.

针对每一个软件缺陷 m_{ij}，执行对应的 5 个均匀随机测试用例，并将缺陷 m_{ij}被测试用例识别的比例 x_{ij}记为缺陷 m_{ij}的暴露概率，以此表征缺陷 m_{ij}的检测难度. 然后将接受评价的测试用例按照其生成的先后顺序在被测试程序上执行，记录暴露概率大于零的软件缺陷 m_{ij}不被在其上执行的前 $k-1$ 个测试用例识别的条件下，被第 k 个测试用例识别的结果 y_{ijk}，简称基于条件的识别结果.

为了使得在第 3 阶段得到的回归模型满足如下 2 个条件：当缺陷的暴露概率为零时，缺陷不可被任意测试用例检测到；当暴露概率为 1 时，缺陷必定可以被任意测试用例检测到，在第 2 阶段，使用式(8)对缺陷的暴露概率 x_{ij}进行预变换，形成 z_{ij}. 进而以变换后的缺陷检测数据 z_{ij}，y_{ijk}，k 作为第 3 阶段回归中的数据样本.

在第 3 阶段，利用自动化建模工具建立 logistic 候选回归模型并进行参数估计. 缺陷不能被第 k 个测试用例识别的条件概率 q_{ijk}往往随着在其上执行的测试用例的执行序号 k 的逐渐增大呈现出以下 4 种主要趋势：逐渐增大，逐渐减小，先减小再增大，先增大再减小. 当 logistic 模型的线性预报器被表示为缺陷 m_{ij}上的测试用例执行序号 k 的二次函数($v=2$)时，logistic 模型可表达上述这 4 种趋势. 于是线性预报器 η_{rs}关于 z_{ij}和 k 的基本形式为 $\eta_{12}(z_{ij},k)$，即

$$\beta_0+\beta_1 z_{ij}+\beta_2 k+\beta_3 k^2+\beta_4 z_{ij}k+\beta_5 z_{ij}k^2.$$

事实上，过于复杂的模型很难从物理意义上进行解释，并且对于现实世界中已知的多数规律，都可以通过建立与之对应的简单模型进行近似[21]. 为了能够在相对简单的模型中，让拟合出的经验回归方程以更大程度接近真实规律，自动化建模工具设置了与基本形式 η_{12} 相对接近的 6 种类型：$\eta_{11}(z_{ij},k)$，$\eta_{12}(z_{ij},k)$，$\eta_{13}(z_{ij},k)$，$\eta_{31}(z_{ij},k)$，$\eta_{32}(z_{ij},k)$，$\eta_{33}(z_{ij},k)$. 对应形成 6 个以 z_{ij}和 k 为自变量的候选 logistic 模型，模型分别记为 $c_{rs}(z,k)$，$r=1,3$，$s=1,2,3$.

为了计算每一个候选 logistic 模型 c_{rs}中的待定系数，自动化建模工具将会调用实现了惩罚极大似然估计方法的 R 语言开源软件包 glmnet 中的 cv.glmnet 函数[22]. 此函数不仅可以对参数系数 β_0，$\boldsymbol{\beta}$ 进行估计，还可获得模型 $c_{rs}(z,k)$ 的经验回归方程的交叉验证得分 v^rs. cv.glmnet 在执行过程中，首先会由大到小产生 100 个惩罚参数 $\lambda_1,\lambda_2\cdots,\lambda_{100}$，其中 λ_1 趋近于正无穷大量，λ_{100}趋近于零，以便在拟合的过程中产生尽可能不同的 100 个经验回归方程，然后 cv. glmnet 从中挑选具有最小交叉验证得分的经验回归方程作为模型 $c_{rs}(z,k)$ 的经验回归方程.

第 4 阶段利用式(8)的反变换对第 3 阶段拟合出的每一个候选 logistic 模型 $c_{rs}(z,k)$ 的经验回归方程 $\hat{q}_{rs}(z,k)$ 进行后变换，得到以缺陷暴露概率 x 为自变量的模型 $c_{rs}(x,k)$ 的经验回归方程 $\hat{q}_{rs}(x,k)$.

第 5 阶段对第 4 阶段得到的每一个候选模型的

经验回归方程 $\hat{q}_{rs}(x,k)$ 进行拟合优度检验. 对于每一个经验回归方程 $\hat{q}_{rs}(x,k)$,自动化建模工具首先建立原假设 H_0:候选模型的经验回归方程 $\hat{q}_{rs}(x,k)$ 能够恰当表达测试用例集合的检错能力. 以经验回归方程 $\hat{q}_{rs}(x,k)$ 为依据,自动化建模工具利用式(18)产生检验统计量 T^* 的 $B=1\,000$ 个随机实现 $T_b, b=1,2,\cdots,B$. 以由小到大排序这 B 个实现. 如果 T_b 位于排序中的第 r 个位置,则将 T_b 记为 $T_{(r)}$. 将 $0.025\times B$ 向上取整记为 l_b,$0.755\times B$ 向下取整记为 u_b. 如果 $T_{\text{obs}}\leqslant T_{(l_b)}$ 或者 $T_{\text{obs}}\geqslant T_{(u_b)}$,说明构成 T_{obs} 的 $|S|$ 个基于条件的缺陷检测结果的发生是小概率发生事件,从而拒绝原假设,丢弃经验回归方程 $\hat{q}_{rs}(x,k)$,否则保留经验回归方程 $\hat{q}_{rs}(x,k)$.

第6阶段进行经验回归方程的选择. 交叉验证得分可以表达经验回归方程的预测误差,从而可以评价模型的相对准确性. 自动化建模工具利用式(16),从第5阶段保留的所有候选经验回归方程 $\hat{q}_{rs}(x,k)$ 中挑选出具有最小交叉验证得分者作为最佳的经验回归方程来表达缺陷被识别的条件概率与缺陷的检测难度以及被测试用例执行的序号之间的联系.

自动化建模工具可以使用式(2)进一步产生缺陷被识别的概率与缺陷的检测难度以及被测试用例执行次数之间的函数关系 $p(x,k)$. 接着,自动化建模工具利用式(3)和式(4)分别计算出被测试程序的每一个表达式的变异评分以及被测试程序的变异评分.

3 自动化建模工具的使用及其实验

使用的被测试程序是一个真实的三角形类型判定程序 tri. c. 测试用例集合包含了由自动化单元测试系统 CTS 针对 tri. c 按照语句覆盖准则产生的6个测试用例. 测试用例按照产生的时间排序为 tc_1、tc_2、tc_3、tc_4、tc_5、tc_6. 自动化建模工具对这个达到语句覆盖标准的测试用例集合的检错能力进行了建模.

在第1阶段,自动化建模工具调用 CTS 系统中的缺陷注入功能,对表达式样本集 S 使用了算数运算符替代、常量替代、关系运算符替代等12类精简的变异算子[20],一共播种了181个变异缺陷,构造了399个基于条件的缺陷检测结果. 每一个基于条件的缺陷检测结果都记录了之前若干个测试用例不能识别该缺陷的条件下缺陷被下一个测试用例识别的结果. 然后自动化建模工具会在缺陷文件上执行测试用例,计算缺陷样本的检测难度.

在第2阶段,自动化建模工具按照式(8)对缺陷样本的暴露概率 x_{ij} 进行预变换,将其转化为 z_{ij}. 在第3节阶段,自动化建模工具调用开源软件包 glmnet 中的 cv. glmnet 函数分别对第2节中描述的6个 logistic 候选模型 $c_{rs}(z,k)$ 的系数 $\beta_0,\boldsymbol{\beta}$ 进行了估计. 在 cv. glmnet 利用惩罚性极大似然估计方法确定模型中的未知系数的过程中采用了10折交叉验证方案,从而得到了每一个 logistic 模型的经验回归方程 $\hat{q}_{rs}(z,k)$.

在 cv. glmnet 利用惩罚性极大似然估计方法确定模型中的未知系数的过程中,cv. glmnet 还提供了每一个经验回归方程 $\hat{q}_{rs}(z,k)$ 的交叉验证得分,用以表达经验回归方程相对准确性.

在第4阶段利用式(8)的反变换对第3阶段拟合出的6个候选 logistic 模型 $c_{rs}(z,k)$ 的经验回归方程 $\hat{q}_{rs}(z,k)$ 进行后变换,得到以缺陷的检测难度 x 和在缺陷上执行测试用例的序号 k 为自变量经验回归方程

$$\hat{q}_{rs}(x,k)=\hat{q}_{rs}[z(x),k]$$

其中 $z(x)=\log[(1-x)/x]$, $r=1,3$, $s=1,2,3$.

在第5阶段,自动化建模工具对第4阶段得到的每一个经验回归方程 $\hat{q}_{rs}(x,k)$ 进行拟合优度检验. 构造拟合优度检验统计量 T^* 的1 000个实现 $T_1,T_2,\cdots,T_{1\,000}$,从而仿真出拟合优度检验统计量 T^* 的概率分布. 最后自动化建模工具以递增的顺序排列此1 000个实现,从而获得拟合优度检验统计量 T^* 的95%置信区间的上界 $T_{(u_b)}$ 和下界 $T_{(l_b)}$.

接着自动化建模工具判断每一个经验回归方程 $\hat{q}_{rs}(x,k)$ 的拟合优度统计量的观测值 T_{obs} 与95%置信区间的上界 $T_{(u_b)}$ 和下界 $T_{(l_b)}$ 之间的关系. 实验数据表明,每一个候选的经验回归方程的拟合优度检验统计量的观测值都处于拟合优度检验统计量的置信区间的上界和下界之间. 因此,自动化建模工具没有淘汰任何一个候选模型的经验回归方程. 自动化建模工具在第6阶段对第5阶段保留的6个候选经验回归方程进行交叉验证得分的比较. 交叉验证得分可以表达经验回归方程的预测误差,从而可以评价模型的相对准确性. 经验回归方程 $\hat{q}_{31}(x,k)$ 的交叉验证的得分在6个经验回归方程中最小,成为

最佳的经验回归方程，最终得到的经验回归方程为

$$\hat{q}(x,k) = \text{logit}^{-1}(0.081 + 0.225z + 0.134z^2 + 0.019z^3 + 0.003k + 0.112zk + 0.014z^2k + 0.018z^3k)$$

其中 $z = \log[(1-x)/x]$. 如图 1 所示，$\hat{q}(x,k)$ 表达了暴露概率为 x 的缺陷不被在其上执行的 $k-1$ 测试用例识别的条件下，依然不被第 k 个测试用例识别的概率.

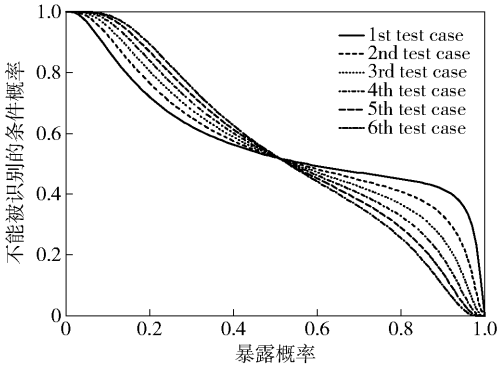

图 1　缺陷不被识别的条件概率 $\hat{q}(x,k)$

自动化建模工具可以进一步产生缺陷被识别的概率与缺陷的检测难度之间的函数关系：

$$p(x,k) = 1 - q(x,1)\cdots q(x,k)$$

这时，表达式被测试用例执行的次数 k 被视为已知量. 最后自动化建模工具将所有表达式样本的变异评分的平均值 0.68 作为被测试程序的变异评分.

4　结束语

给出了一种基于缺陷检测难度建模测试用例集合的检错能力的方法. 这种方法不但能够表达缺陷被检测到的概率与缺陷检测难度之间的数量关系，还能够表达测试用例按照其产生时间的先后顺序呈现出的检错能力的变化趋势. 重新定义的被测试程序的变异评分规避了经典变异评分的计算过程中因被播种的缺陷集的检测难度的变化而使得变异评分的计算不稳定的缺点. 目前的自动化建模工具的建模对象只能是由数值运算构成的单个被测试函数. 下一步的工作重点是扩大自动化建模工具的适用范围，使得自动化建模工具可以适用于包含有函数调用以及字符串、结构体等复杂数据结构的 C 语言被测试程序.

参考文献：

[1] Zhu Hong, Hall P A V, May J H R. Software unit test coverage and adequacy [J]. ACM Computing Survey, 1997, 29(4): 366-427.

[2] Hamlet R G. Testing programs with the aid of a compiler [J]. IEEE Transactions on Software Engineering, 1977, 3(4): 279-290.

[3] DeMillo R A, Lipton R J, Sayward F G. Hints on test data selection: help for the practicing programmer [J]. Computer, 1978, 11(4): 34-41.

[4] Andrews J H, Briand L C, Labiche Y. Is mutation an appropriate tool for testing experiments? [C]// Proceedings of the 27th International Conference on Software Engineering (ICSE'05). New York: ACM, 2005: 402-411.

[5] Do H, Rothermel G. On the use of mutation faults in empirical assessments of test case prioritization techniques [J]. IEEE Transactions on Software Engineering, 2006, 32(9): 733-752.

[6] Offutt A J, Lee A, Rothermel G, et al. An experimental determination of sufficient mutant operators [J]. ACM Transactions on Software Engineering and Methodology, 1996, 5(2): 99-118.

[7] Jia Yue, Harman M. Constructing subtle faults using higher order mutation testing [C]// Proceedings of the 2008 8th IEEE International Working Conference on Source Code Analysis and Manipulation. Beijing: IEEE, 2008: 249-258.

[8] DeMillo R A, Guindi D S, McCracken W M, et al. An extended overview of the Mothra software testing environment[C]// Proceedings of the second Workshop on Software Testing, Verification, and Analysis. Banff: IEEE, 1988.

[9] Delamaro M, Maldonad J C. Proteum-A tool for the assessment of test adequacy for C programs[C]// Proceedings of the Conference on Perform Ability in Computing Systems. Brunswick, NJ: [s. n.], 1996: 79-95.

[10] Offutt A J, Hayes J H. A semantic model of program faults[C]// Proceedings of the 1996 ACM SIGSOFT International Symposium on Software Testing and Analysis. San Diego: ACM, 1996: 195-200.

[11] DeMillo R A, Offutt A J. Constraint-based automatic test data generation[J]. IEEE Transactions on Software Engineering, 1991, 17(9): 900-910.

[12] Voas J M. PIE: a dynamic failure-based technique [J]. IEEE Transactions on Software Engineering, 1992, 18(8): 717-727.

[13] Chen W, Untch R H, Rothermel G, et al. Can fault-exposure-potential estimates improve the fault detection a-

bilities of test suites? [J]. Software Testing, Verification and Reliability, 2002, 12 (4): 197-218.

[14] Tervonen T, van Valkenhoef G, Bastürk N, et al. Hit-and-run enables efficient weight generation for simulation-based multiple criteria decision analysis[J]. European Journal of Operational Research, 2013, 224(3): 552-559.

[15] Valkenhoef G V, Tervonen T, Postmus D. Notes on Hit-and-run enables efficient weight generation for simulation-based multiple criteria decision analysis[J]. European Journal of Operational Research, 2014, 239(3): 865-867.

[16] Harrell F E. Regression modeling strategies with applications to linear models, logistic regression and survival analysis [M]. New York: Springer, 2001.

[17] Cessie S L, Houwelingen J C V. Ridge estimators in logistic regression [J]. Applied Statistics, 1992, 41 (1): 191-201.

[18] Gelman A, Hill J L. Data analysis using regression and multilevel/hierarchical models [M]. Cambridge: Cambridge University Press, 2006.

[19] 金大海, 宫云战, 王雅文, 等. 软件代码测试技术[J]. 信息通信技术, 2015(3): 34-39.
Jin Dahai, Gong Yunzhan, Wang Yawen, et al. Software code test technology[J]. Information and Communications Technologies, 2015(3): 34-39.

[20] 钱莨南, 宫云站, 王雅文, 等. 面向C语言的故障注入平台[J], 北京邮电大学学报, 2016, 39(3): 95-99.
Qian Gennan, Gong Yunzhan, Wang Yawen, et al. A C-language oriented fault injection platform[J]. Journal of Beijing University of Posts and Telecommunications, 2016, 39(3): 95-99.

[21] Mirman D. Growth curve analysis and visualization using R [M]. Boca Raton, FL: Chapman & Hall/CRC Press, 2014.

[22] Friedman J, Hastie T, Tibshirani R. Regularization paths for generalized linear modelsvia coordinate descent [J]. J Stat Softw, 2010(33): 1-22.