TBFL和SAFL方法熵分析

王蓁蓁[1,2] 徐宝文[3,4] 周毓明[3,4] 陈 林[3,4]
(金陵科技学院信息技术学院 南京 211169)[1] (江苏省信息分析工程实验室 南京 211169)[2]
(南京大学软件新技术国家重点实验室 南京 210093)[3] (南京大学计算机科学与技术系 南京 210093)[4]

摘 要 为TBFL(testing-based fault localization)方法和SAFL(similarity-aware fault localization)方法构造了熵模型,并用该模型对Dicing方法、TARANTULA方法、SAFL方法在一个实例上进行分析比较。结果表明,熵模型可以为构造以及分析TBFL方法和SAFL方法提供一个原则性框架。

关键词 熵,错误定位,相似性错误定位
中图法分类号 TP311 **文献标识码** A

Entropy Analysis of Testing-based Fault Localization and Similarity-aware Fault Localization

WANG Zhen-zhen[1,2] XU Bao-wen[3,4] ZHOU Yu-ming[3,4] CHEN Lin[3,4]
(School of Information Technology, Jinling Institute of Technology, Nanjing 211169, China)[1]
(Information Analysis Engineering Laboratory of Jiangsu Province, Nanjing 211169, China)[2]
(State Key Laboratory of Novel Software Technology, Nanjing University, Nanjing 210093, China)[3]
(Department of Computer Science and Technology, Nanjing University, Nanjing 210093, China)[4]

Abstract We presented an entropy model for the TBFL(testing-based fault localization) approaches and SAFL(similarity-aware fault localization) approach. We used the model to compare and analyze the Dicing approach, TARANTULA approach and SAFL approach on one instance. The analysis demonstrates that the entropy model can not only provide a new type of TBFL, but also provide a principle framework for constructing and analyzing various TBFL approaches and SAFL approach.

Keywords Entropy, Testing-based fault localization, Similarity-aware fault localization

1 引言

寻找软件错误位置是一件困难和费时的工作,但它关系到软件质量的提高,所以许多年来,软件界从不同的角度开发了许多方法,用以有效地解决错误定位问题。其中运用从软件测试里获得的信息自动确定错误位置是很重要的一种技术手段,这些方法可以统称为TBFL(testing-based fault localization)方法[1-3]。TBFL基于许多测试用例对软件测试的结果,从中挑选出一些值得怀疑的语句,即它们可能包含软件错误,并按怀疑程度进行排序,从而帮助开发人员缩小搜索范围,尽可能快地找到错误根源,提高开发质量。

文献[3]讨论了6种涉及动态定位的TBFL方法:Dicing方法(Agrawl et al. 1995)、TARANTULA方法(Jones et al. 2002;Jones and Harrold 2005)、Nearest Neighbor Queries方法(Renieris and Reiss 2003)、CT(Cleve and Zeller 2005)、SOBER(Liu et al,2005)和liblitas(liblit et al. 2005),发现它们都忽略了实施相似性的测试用例可能产生的问题。测试用例之间的相似性是指它们都覆盖若干个相同语句,其极端情况便是冗余,即测试用例由完全相同的语句覆盖。文献[6]和文献[3]指出,相似性的测试用例可能会损害TBFL方法的功效。为此,文献[3]提出了SAFL(similarity-aware fault localization)方法,它把每一个测试用例都看作是一个Fuzzy集合,并用概率理论计算语句的怀疑程度,借此处理测试用例之间的相似性问题,有效避免了相似性对错误定位的副作用。文献[3]通过两类实验比较SAFL方法、Dicing方法、TARANTULA方法的功效,得出:SAFL方法不仅能够有效地处理包含了许多冗余测试用例的测试集,而且在没有多少冗余测试用例的测试用例集上也能有效地执行。

在测试实践方面,测试用例之间的相似性总是难以避免的,所以讨论类似SAFL方法是有价值的。信息论是关于通信过程本质最深刻的理论,自1948年Claude Shannon的开创性工作以来,它已经为研究信息根本问题提供了一个总体框架,也为信息处理系统的设计提供了标准。上述TBFL、SAFL等方法也可看作是对测试用例实施结果中的信息(即

到稿日期:2013-01-03 返修日期:2013-04-30 本文受国家自然科学基金重大研究计划重点项目(90818027),国家自然科学基金面上项目(60773104,60803007),金陵科技学院科研基金(jit-b-201207)资助。

王蓁蓁(1975—),女,博士后,主要研究方向为软件测试、人工智能,E-mail:wangzhenzhen@seu.edu.cn;徐宝文(1961—),男,博士,教授,博士生导师,主要研究方向为程序设计语言、软件工程等;周毓明(1974—),男,博士,教授,主要研究方向为软件度量;陈 林(1979—),男,博士,主要研究方向为程序分析。

从中提取错误语句的信息)进行处理的方法,因此本文将以一种原则性方式开发关于 TBFL 和 SAFL 方法的熵论模型,其主要目的是证实文献[3]的结论,并为更好地设计其他 TBFL 和 SAFL 方法提供理论框架和标准。

为了方便,今后如无特殊声明,也把相似性用例称为冗余用例,即互用冗余性和相似性两个概念。同样地,也把文献[3]提出的 SAFL 方法归到 TBFL 方法中,即当说到 TBFL 方法时,也认为它包括了 SAFL 方法。

本文第 2 节预备知识,主要介绍熵的基本概念;第 3 节介绍熵模型;第 4 节是实例分析:主要利用熵模型对几个 TBFL 方法进行熵分析;最后是结论和展望。

2 预备知识

遵循概率论记法,用大写字母(例如 X)表示随机变量,小写字母(例如 x)表示随机变量的值,并且本文只讨论有限离散随机变量。

下面讨论的基本概念可以从任何涉及到信息论知识或其应用的有关书籍中找到,例如可以参看神经网络原理[4]第 10 章,第 2 节(P. 352-P. 353)。

定义 1(熵) 设有限离散随机变量 $X=\{x_k \mid k=1,\cdots,m\}$,其中 $x_k(k=1,2,\cdots,m)$ 是 X 的一个离散值。假设 X 取每一个离散值 x_k 的概率为 $p_k=P(X=x_k)$,它们满足下述两个条件:

$$0\leqslant p_k\leqslant 1,\sum_{k=1}^{m}p_k=1$$

用 $H(X)$表示 X 的熵,由下式

$$H(X)=-\sum_k p_k\log p_k \tag{1}$$

定义,其中对数可以取任意值为底,并且 $0\cdot\log 0=0$。

如果把 X 的每一个实现(即 $X=x_k$)看作一个消息,那熵 $H(X)$就表示每一条消息所携带的信息的平均量。容易证明:

$$0\leqslant H(X)\leqslant\log m \tag{2}$$

其中,m 是 X 取的离散值总数。

特别重要的是,熵 $H(X)$表示 X 的不确定性,下面例题可以直观说明这一点。

例题:(1)$H(X)=0$ 当且仅当对于某一 x_k,X 取 x_k 的概率 $p_k=1$,而对 X 的其他取值,其概率皆为 0。熵的这个下界说明 X 没有不确定性,或者说它的不确定性为 0,因为它取 x_k 值是确定的。

(2)$H(X)=\log m$ 当且仅当对所有的 x_k,$k=1,2,\cdots,m$,X 取 x_k 的概率皆相等,即 $p_k=\frac{1}{m}$。熵的这个上界对应最大不确定性。确实如此,因为这时我们对 X 几乎没有什么先验知识,不能确定它的取值。

3 熵模型

(1)程序随机变量 X

程序用语句集合表示,即用

$$X=\{x_1,x_2,\cdots,x_m\}$$

表示一个程序,其中 $x_i(i=1,2,\cdots,m)$表示程序 X 的第 i 个(可以执行)语句。

程序的错误来源于它的语句错误。TBFL 方法的要旨就是根据测试结果重新估计每一条语句发生错误的可能性,从而帮助开发人员迅速找出错误语句进行修正。鉴于这种考虑,我们认为每个语句都有可能导致程序失败,根据编程方面的理论和实践,可以在测试之前先估计各个语句(或语句类型)的出错概率,记为 $r_i=p(x_i)$,$i=1,2,\cdots,m$。如果没有这方面的资料可循,可以根据统计中"同等无知"原则,认为 $r_i=\frac{1}{m}$,$i=1,2,\cdots,m$。这样,把程序抽象成随机变量 X,它取(可以执行)的语句为值,而且知道程序在每一个语句出错的概率。于是根据式(1),求出程序 X 的熵为:

$$H(X)=-\sum_k p_k\log p_k \tag{3}$$

$H(X)$表示程序 X 在错误语句的定位上的先验不确定性。

(2)TBFL 方法的熵模型

给定 $X=\{x_1,x_2,\cdots,x_m\}$,假定用测试集 $T=\{t_1,t_2,\cdots,t_n\}$进行测试。其中 $t_j(j=1,2,\cdots,n)$表示测试用例,如果 t_j 的测试结果与预想结果不一致,称 t_j 为失败用例;如果 t_j 的测试结果与预想结果一致,称 t_j 为通过测试。所有的失败测试用例组成的集合用 T_f 表示,所有通过测试用例的集合用 T_p 表示。T_f 和 T_p 都是测试用例集合 T 的子集,即 $T=T_f\cup T_p$。

TBFL 方法是根据测试集 T 的测试结果对 X 中语句的出错可能性进行重新估计。换言之,是在 T 的条件下计算原程序语句出错的概率。可以形象地用 $X\mid T$ 表示一个新随机变量,它的取值仍然为 $X\mid T=\{x_1,x_2,\cdots,x_m\}$,不过取每一值的概率与原程序随机变量 X 取值的概率有所不同。为了避免混淆,我们用 Y 记这个随机变量 $X\mid T$,用 $p'(y)$表示 Y 的概率分布,不过 Y 取值和 X 的取值相同,即 $Y=\{y_1,y_2,\cdots,y_m\}$,而 $y_i=x_i$,$i=1,2,\cdots,m$。

用 $H(Y)$表示随机变量 Y 的熵,即

$$H(Y)=-\sum_k p'(y_k)\log p'(y_k) \tag{4}$$

它表示利用测试用例结果,我们保留的原程序 X 的不确定性。

既然 $H(X)$是程序 X 的先验不确定性,$H(Y)$是观测到 T 的结果后(重新估计的)新变量 $X\mid T$ 的不确定性,即 $H(Y)$保留的原程序的不确定性,故 $H(X)-H(Y)$的差就可以表示在观测到 T 的结果后程序不确定性的减少量,或者说程序确定性的增加量。用 $M(X;Y)$表示这个改变量,即

$$M(X;Y)=H(X)-H(Y) \tag{5}$$

称 $M(X;Y)$为 X 在 T 条件下确定性的增益,简称为增益。

(3)模型的应用

主要应用有两个方面:

· 固定 X。假设向无(或较少)冗余测试集 T^* 增添相似性用例后得到测试集 T。设 TBFL 方法 L 在 T 和 T^* 上观察到的变量分别为 Y 和 Y^*。令

$$S=H(Y)-H(Y^*);\delta=M(X;Y)-M(X;Y^*) \tag{6}$$

如果 $S>0$,则 L 在冗余测试集上保留的不确定性反而增多;$\delta<0$,则 L 在冗余测试集上得到的确定性反而减少。

这时就说明 L 方法在冗余测试集上不能有效解决冗余可能引起的损害 L 方法功效的问题。

· 固定 X 和 T。设 L_1,L_2 是两个不同的 TBFL 方法,

Y_1, Y_2 是 L_1, L_2 通过测试集 T 分别得到的新变量。令

$$S=H(Y_1)-H(Y_2);\delta=M(X;Y_1)-M(X;Y_2) \quad (7)$$

如果 $S>0$，则 L_1 保留的不确定性较多；$\delta<0$，则 L_1 获得的确定性较少。于是 L_1 方法在错误定位的总性能方面(平均来说)较 L_2 方法差。

4 实例分析

我们用图 1(a faulty program and its execution traces)中给出的程序和测试用例，用熵模型对 Dicing、TARANTULA、SAFL 3 个方法进行分析。有关该图的详细情况可查阅文献[3]。

		Test suite 1				Test suite 2			
		t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
Mid ( ) { int x, y, z, m;	statements	2,1,3	1,3,2	5,5,3	1,2,3	3,3,5	1,1,2	2,2,3	5,5,2
read ("Enter 3 numbers:", x, y, z);	x_1	●	●	●	●	●	●	●	●
m = x;	x_2	●	●	●	●	●	●	●	●
if (y < z)	x_3	●	●	●	●	●	●	●	●
if (x < y)	x_4	●			●	●	●	●	
m = y;	x_5				●				
else if (x < z)	x_6	●				●	●	●	
m = y;	x_7	●				●	●	●	
else	x_8		●	●					●
if (x > y)	x_9		●	●					●
m = y;	x_{10}								
else if (x > z)	x_{11}		●	●					●
m = x;	x_{12}			●					●
printf ("Middle number is:" m);	x_{13}	●	●	●	●	●	●	●	●
}		F	F	P	P	P	P	P	P

图 1 A faulty program and test suites

4.1 程序 X，测试集 T 和 T^*

(1)程序 X

现在，程序 $X=\{x_1, x_2, \cdots, x_{13}\}$，其中 x_i 表示程序编号为 i 的语句，例如 $x_1=$read("Enter 3 numbers:" x, y, z)。如果事先并不知道每个语句出错的可能性，可以假定每个语句出错的概率为 $p_i=p(x_i)=\frac{1}{13}, i=1,2,\cdots,13$。下面我们就做这样的假定。于是按式(3)求出 X 的熵 $H(X)$为：

$$H(X)=-\sum_i p_i \log p_i=\log 13=1.1139433 \quad (8)$$

其中对数取 10 为底(下面计算对数时皆以 10 为底)。

(2)测试集 T

选择 Test suite 1＋Test suite 2 作为整个测试集 T。T 有 8 个用例，按图 1 顶格右边给出的次序分别把它们编号为 $t_1, t_2, \cdots, t_8$。例如 t_2 是用例：$x=1, y=3, z=2$。在 8 个用例中，t_1, t_2 是失败用例，其余 6 个是通过用例。用 T_f, T_P 分别表示失败用例集合和通过用例集合，即

$$T_f=\{t_1, t_2\}, T_P=\{t_3, \cdots, t_8\}$$

(3)测试集 T^*

选择 Test suite 1 作为无冗余测试集 T^*，T^* 只有 4 个用例，和上面的编号一致，$T^*=\{t_1, t_2, t_3, t_4\}$，其中 $t_1, t_2 \in T_f^*$，$t_3, t_4 \in T_P^*$。

显然，T 是通过增添 t_5, t_6, t_7, t_8 等 4 个冗余用例到 T^* 后形成的。

下面以上述 X, T 和 T^* 对几个 TBFL 方法进行熵分析，由于用到文献[3]中的资料表 1(Results of the motivating example)，为了方便读者，也把它列于表 1。有关表 1 的详细情况参阅文献[3]。

表 1 Results of Dicing approach and TARANTULA approach

The results of test suite 1												
Statement	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}
Dicing	0	0	0	1	0	2	2	1	1	0	1	0	0
TARANTULA	0.5	0.5	0.5	0.5	0	1	1	0.5	0.5	—	0.5	0	0.5
The results of test suite 1＋test suite 2													
Statement	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}
Dicing	0	0	0	2	0	3	3	4	4	0	4	0	0
TARANTULA	0.5	0.5	0.5	0.43	0	0.5	0.5	0.6	0.6	—	0.6	0	0.5

4.2 Dicing 方法熵分析

这里分析的 Dicing 方法是基于原 Dicing 方法的推广[3]。Dicing 方法是寻找语句集合 dice，它是从一个失败的测试用例运行时所覆盖的语句集合(记为 S_f)中减去一个通过测试用例运行时所覆盖的语句集合(记为 S_p)构成的，即 $dice=S_f-S_p$。求出所有的 dice 以后，对每一语句计算它被 dice 所包含的 dice 个数，根据个数的大小排序。语句属于 dice 的个数越大，它就越值得怀疑，即它出错的可能性越高。

(1)测试集中包含冗余用例的情况

根据测试集 T 的测试结果，Dicing 方法对语句的怀疑程度进行排序，见表 1。我们利用这个排序规定新随机变量 $Y=X|T$ 的分布($p'(y)$)，如表 2 所列。

表 2 $Y=X|T$ 在 Dicing 方法中的分布

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}
Dicing 排序	0	0	0	2	0	3	3	4	4	0	4	0	0
$p'(y)$	0	0	0	1/10	0	3/20	3/20	1/5	1/5	0	1/5	0	0

这里 $Y=X|T$ 表示利用测试结果对语句出错概率重新调整后按新的分布变化的随机变量，新的概率质量函数记为 $p'(y_i), y_i=x_i, i=1,2,\cdots,13$。

按式(4)，求出 $H(Y)$：

$$\begin{aligned} H(Y)&=-\sum_k p'(y_k)\log p'(y_k)=-\sum_k p'(x_k)\log p'(x_k) \\ &=-[\frac{1}{10}\log\frac{1}{10}+(\frac{3}{20}\log\frac{3}{20})\times 2+(\frac{1}{5}\log\frac{1}{5})\times 3] \\ &=0.7665546 \end{aligned} \quad (9)$$

它表示 $X|T$ 中保留 X 的不确定性。

再按式(5)、式(8)和式(9)，求出 $M(X;Y)$：

$$\begin{aligned} M(X;Y)&=H(X)-H(Y)=1.1139433-0.7665546 \\ &=0.3473887 \end{aligned} \quad (10)$$

它表示实施测试后，利用测试集 T 获得的确定性增益。

(2)测试集中无冗余用例的情况

根据测试集 T^* 的测试结果，Dicing 方法对语句的怀疑程度进行排序，见表 1。我们利用这个排序规定新随机变量 $Y^*=X|T^*$ 的分布，如表 3 所列。

表 3 随机变量 $Y^*=X|T^*$ 在 Dicing 方法中的分布

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}
Dicing 排序	0	0	0	1	0	2	2	1	1	0	1	0	0
$p'(y)$	0	0	0	1/8	0	1/4	1/4	1/8	1/8	0	1/8	0	0

同理，计算 $H(Y^*)$和 $M(X;Y^*)$如下：

$$H(Y^*)=-\sum_k P'(y_k)\log P'(y_k)=0.7525748 \quad (11)$$

$$M(X;Y^*)=H(X)-H(Y^*)=0.3613685 \quad (12)$$

它们分别表示利用测试集 T^* 的测试结果，保留原程序不确定性以及确定性增益的情况。

(3)冗余测试对功效伤害程度的度量

按式(6)、式(9)、式(11)，计算 S：

$$S=H(Y)-H(Y^*)=0.7665546-0.7525748=0.0139798 \quad (13)$$

按式(6)、式(10)、式(12)，计算 δ：

$$\delta=M(X;Y)-M(X;Y^*)=0.3473887-0.3613685=-0.0139798 \quad (14)$$

因为 $S>0$，$\delta<0$，可见当向无冗余测试用例集 T^* 上增添冗余测试用例后，在测试用例集 T 上，Dicing 方法的功效受到伤害。

4.3 TRANTULA 方法熵分析

Jones 和 Harrold 在文献[7](2005)采用下列公式表达对语句的怀疑程度[3]：

$$\text{suspiciousness}(i)=1-\text{hue}(i)=\frac{\%\text{failed}(i)}{\%\text{passed}(i)+\%\text{failed}(i)}$$

式中，%passed(i)是覆盖语句 i 的通过测试的用例个数与所有通过测试的用例个数之比；%failed(i)是覆盖语句 i 的失败测试用例个数与所有失败测试用例个数之比。怀疑程度高的语句，其出错的可能性大。TARANTULA 方法按怀疑程度对语句进行排序。

(1)测试集中包含冗余用例的情况

根据测试集 T 的测试结果，TARANTULA 方法计算语句怀疑程度，见表 1。我们利用这个结果规定新随机变量 $Y=X|T$ 的概率分布，用 $p'(y_k)$记之，其中 $y_k=x_k$，$k=1,2,\cdots,13$，如表 4 所列，其中“—”表示语句未执行。我们可以把未执行语言的概率作为 0 处理，N 是归一化因子，由 $\sum_k p'(y_k)=1$，可得 $N=\frac{1}{5.23}$，从而可得 Y 的概率分布，例如当 $Y=x_6$ 时，$p'(x_6)=0.5N=\frac{0.5}{5.23}=\frac{5}{52.3}$。

表 4　$Y=X|T$ 在 TARANTULA 方法下的分布

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}
TARANTULA	0.5	0.5	0.5	0.43	0	0.5	0.5	0.6	0.6	—	0.6	0	0.5
p'(y)	0.5N	0.5N	0.5N	0.43N	0	0.5N	0.5N	0.6N	0.6N	0	0.6N	0	0.5N

按式(4)，计算 $H(Y)$：

$$H(Y)=-\sum_k p'(y_k)\log p'(y_k)=-\sum_k p'(x_k)\log p'(x_k)=0.9976653 \quad (15)$$

它表示通过观测 T 的结果后，保留 X 的不确定性。

按式(5)、式(8)和式(15)，求出 $M(X;Y)$：

$$M(X;Y)=H(X)-H(Y)=0.116278 \quad (16)$$

它表示实施测试后，利用测试集 T 获得的确定性增益。

(2)测试集中无冗余测试用例的情况

根据测试集 T^* 的测试结果，TARANTULA 方法计算语句怀疑程度，见表 1。利用这个结果规定随机变量 $Y^*=X|T^*$ 的概率分布 $p'(y_k)$，$y_k=x_k$，$k=1,2,\cdots,13$，如表 5 所列。

表 5　$Y^*=X|T^*$ 在 TARANTULA 方法下的分布

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}
TARANTULA	0.5	0.5	0.5	0.5	0	1	1	0.5	0.5	—	0.5	0	0.5
p'(y)	0.5N	0.5N	0.5N	0.5N	0	N	N	0.5N	0.5N	0	0.5N	0	0.5N

其中“—”表示语句 x_{10} 未执行，但我们也把它作为 0 处理。N 是归一化因子，由 $\sum_k p'(y_k)=1$，可得 $N=\frac{1}{6}$。于是可求出 Y^* 的分布，例如 $p'(y_6)=p'(x_6)=\frac{1}{6}$。

同理，计算 $H(Y^*)$和 $M(X;Y^*)$如下：

$$H(Y^*)=-\sum_k p'(y_k)\log p'(y_k)=0.9788378 \quad (17)$$

$$M(X;Y^*)=H(X)-H(Y^*)=0.1351055 \quad (18)$$

它们分别表示通过测试集 T^* 的测试结果，保留原程序 X 的不确定性和确定性增益情况。

(3)冗余测试对功效伤害程度的度量

按式(6)、式(15)、式(17)，计算 S：

$$S=H(Y)-H(Y^*)=0.0188275 \quad (19)$$

按式(6)、式(16)、式(18)，计算 δ：

$$\delta=M(X;Y)-M(X;Y^*)=-0.0188275 \quad (20)$$

因为 $S>0$，$\delta<0$，可见当向测试集 T^* 里增加冗余测试用例后，在测试用例集 T 上，TARANTULA 方法的功效的确受到伤害。

4.4 SAFL 方法熵分析

Hao 所提出的 SAFL 方法首先记录下测试用例的执行结果。如果程序有 m 个语句，测试集 T 有 n 个用例。用 T 对 X 进行测试，将测试结果记录成 $n\times(m+1)$阶矩阵，它称为执行矩阵，记作 $E=(l_{ij})$，$i=1,2,\cdots,n$；$j=1,2,\cdots,m,m+1$。其中

$$l_{ij}=\begin{cases}1, & \text{用例 } t_i \text{ 覆盖语句 } x_j\ (1\leqslant j\leqslant m)\\ 1, & t_i \text{ 是通过测试}\ (j=m+1)\\ 0, & \text{其他情况}\end{cases}$$

换言之，E 中第 i 行记录了测试用例 t_i 的执行情况、它所覆盖的语句(即 $l_{ij}=1(1\leqslant j\leqslant m)$的语句 x_j)以及用例的类型(即 $l_{im+1}=1$，若 $t_i\in T_P$；$l_{im+1}=0$，若 $t_i\in T_f$)。

然后将每一个测试用例看作一个 Fuzzy 集合，例如将用例 t_i 看作：

$$\tilde{t}_i=\{x_1/f_{i1},x_2/f_{i2},\cdots,x_m/f_{im}\}$$

式中，f_{ij} ($j=1,2,\cdots,m$)是语句 x_j 的隶属度(即 $f_{ij}=\mu_{\tilde{t}_i}(x_j)$)，它由下式计算：

$$f_{ij}=\begin{cases}1/\sum_{k=1}^{m} l_{ik}, & \text{如果 } l_{ij}=1\\ 0, & \text{其他情况}\end{cases}$$

由所有的用例 Fuzzy 集合组成 Fuzzy 矩阵 $\tilde{F}$，即

$\tilde{F}=(f_{ij})$，$1\leqslant i\leqslant n$，$1\leqslant j\leqslant m$。

对于每一个语句，例如 x_j 构造 $F-j$ 和 $A-j$。$F-j$ 是所有覆盖 x_j 的失败用例 t_i(即 $l_{ij}=1$，$l_{im+1}=0$)的 Fuzzy 集合 $\tilde{t}_i$ 的并集。$A-j$ 是所有覆盖语句 x_j 的用例 t_i(即 $l_{ij}=1$)的 Fuzzy 集合 $\tilde{t}_i$ 的并集。用符号 $|F-j|$ 和 $|A-j|$ 分别表示 Fuzzy 集 $F-j$ 和 $A-j$ 的基数。简略地说，若干个 Fuzzy 集的并集仍是一个 Fuzzy 集，其成员隶属度是诸 Fuzzy 集相应成员的隶属度的最大值，而一个 Fuzzy 集的基数是该集所有成员的隶属度之和[3]。

有了上面的信息即可定义语句的怀疑度。例如 x_j 语句的怀疑度用 $P(j)$表示，它由下式计算：

$$P(j)=\frac{|F-j|}{|A-j|}$$

最后根据 $P(j)$对语句排序，$P(j)$越大，它的排序越靠前。

现在我们用熵模型来分析 SAFL 方法。

(1)测试用例集中包含冗余用例的情况

$X=\{x_1,\cdots,x_{13}\}$，$T=\{t_1,\cdots,t_8\}$，$p(x_i)=\frac{1}{13}$，$i=1,2,\cdots,13$

根据测试集 T，由文献[3]表 2(P. 223)计算 $p(j)$的结果，我们规定 $Y=X|T$ 的概率分布 $p'(j)$，如表 6 所列。

表 6　$Y=X|T$ 在 SAFL 方法下的分布

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}
SAFL 的 P(j)	0.78	0.78	0.78	0.78	0	1	1	0.89	0.89	0	0.89	0	0.78
p′(j)	0.78N	0.78N	0.78N	0.78N	0	N	N	0.89N	0.89N	0	0.89N	0	0.78N

其中 N 是归一化因子，由 $\sum_j p'(j)=1$，得到 $N=\frac{1}{8.57}=\frac{100}{857}$。于是我们得到 $Y=X|T$ 随机变量的分布，例如 Y 取 x_6 的概率为 $p'(6)=\frac{1}{8.57}=\frac{100}{857}$。

和上面算法一样，可得：

$$H(Y)=-\sum_j p'(j)\log p'(j)=0.9978535 \quad (21)$$

它表示观测到 T 结果后，保留 X 的不确定性指标。

$$M(X;Y)=H(X)-H(Y)=0.1160898 \quad (22)$$

它表示观测到 T 结果后，对 X 的确定性的增益。

(2)测试集中无冗余用例的情况

由文献[3]表 2(P. 223)看出，根据测试集 T^* 计算出的 $p(j)$和根据测试集 T 计算出的 $p(j)$完全一样，所以 $Y^*=X|T^*$ 的概率分布和 $Y=X|T$ 的概率分布也完全一样。因此

$$H(Y^*)=0.9978535 \quad (23)$$

$$M(X;Y^*)=0.1160898 \quad (24)$$

它们也分别和 $H(Y)$、$M(X;Y)$完全一样。

(3)冗余测试对功效伤害程度的度量

$$S=H(Y)-H(Y^*)=0 \quad (25)$$

$$\delta=M(X;Y)-M(X;Y^*)=0 \quad (26)$$

因为 $S=0$，$\delta=0$，所以 SAFL 方法确实避免了因冗余测试而引起的功效损害问题。

4.5　3 种方法的对比

(1)测试集中有冗余测试情况

根据上面的分析，对于 Dicing 方法，有

$H(Y)=0.7665546$，$M(X;Y)=0.3473887$

对于 TARANTULA 方法，有

$H(Y)=0.9976653$，$M(X;Y)=0.116278$

对于 SAFL 方法，有：

$H(Y)=0.9978535$，$M(X;Y)=0.1160898$

由于 Y 是表示在观察到测试用例结果后的程序随机变量，那么根据信息论，随机变量 Y 的熵表示所保留的原程序的不确定性。因此在保持不确定性方面，有：

Dicing<TARANTULA<SAFL

同样，既然 $H(X)$是程序 X 的先验不确定性，$H(Y)$是观测到 T 的结果后(重新估计的)新变量的不确定性，即 $H(Y)$保留的原程序的不确定性，故 $H(X)-H(Y)$的差就可以表示在观测到 T 的结果后程序不确定性的减少量，或者说程序确定性的增加量，也就是 $M(X;Y)$。因此，在增添确定性方面，有：

SAFL<TARANTULA<Dicing

所以在有冗余测试时总的性能方面：

SAFL<TARANTULA<Dicing

但是 SAFL 几乎与 TARANTULA 性能一样，只差 0.01%～0.02%。

(2)测试集中无冗余用例的情况

同样，根据上面的分析，对于 Dicing 方法，有

$H(Y^*)=0.7525748$，$M(X;Y^*)=0.3613685$

对于 TARANTULA 方法，有

$H(Y^*)=0.9788378$，$M(X;Y^*)=0.1351055$

对于 SAFL 方法，有

$H(Y^*)=0.9978535$，$M(X;Y^*)=0.1160898$

Y 是表示在观察到测试用例结果后的程序随机变量，那么根据信息论，随机变量 Y 的熵表示所保留的原程序的不确定性。因此在保持不确定性方面，有：

Dicing<TARANTULA<SAFL

$H(X)$是程序 X 的先验不确定性，$H(Y)$是观测到 T 的结果后(重新估计的)新变量的不确定性，即 $H(Y)$保留的原程序的不确定性，故 $H(X)-H(Y)$的差就可以表示在观测到 T 的结果后程序不确定性的减少量，或者说程序确定性的增加量，即 $M(X;Y)$。因此，在增添确定性方面，有：

SAFL<TARANTULA<Dicing

所以在无冗余测试时总的性能方面：

SAFL<TARANTULA<Dicing

但是在这种情况下，SAFL 与 TARANTULA 的相差比有冗余测试时增大了些，现在相差提高到 1% ～ 2%。这是不难理解的，因为 SAFL 是专门为消除冗余测试可能在定位上的伤害而设计的，所以在无冗余测试时，总性能下降是难免的。

(3)对冗余可能伤害的避免性能情况

根据前面的分析，对于 Dicing 方法，有：

$S=0.013798$，$\delta=-0.0139798$

关于 TARANTULA 方法，有：

$S=0.0188275$，$\delta=-0.0188275$

关于 SAFL 方法，有：

$S=0$，$\delta=0$

从 T^* 到 T，保持不确定性方面的比较：

SAFL<Dicing<TARANTULA

从 T^* 到 T，在增添确定性方面的比较：

TARANTULA<Dicing<SAFL

从 T^* 到 T，性能改变方面的比较：

TARANTULA<Dicing<SAFL

当向无冗余测试集增添冗余测试用例时，就错误定位而言，TARANTULA 和 Dicing 都可能受到伤害，但 SAFL 确实达到了设计它的目标能够避免这种类型的伤害，而且在性能方面比 Dicing 还高 1%～2%。

结论：无论是有冗余测试或是无冗余测试，都以 Dicing 方法的性能较好，然而这 3 个方法的区别并不很大，尤其是 SAFL 和 TARANTULA 相差更是微小，可以忽略不计。但是当向测试集中增添更多用例时，在实践中很难避免这些测试之间的相似性，这时在对冗余测试可能伤害错误定位的功效这个意义上，SAFL 表现的优越性是显著的。这里的结果与文献[3]在做了两类实验后所得的结果基本上是一致的。显然程序 X、测试集 T 和 T^* 都很小且不具有一般性，但是在它们之上对 TBFL 方法进行分析，却显示出我们的熵模型确

实在分析 TBFL 方法性能方面有重要作用。

结束语 运用测试集对程序错误语句定位问题，除了上述所提及的几个主要方法，许多学者提出了许多其它行之有效的方法。但大多冗余测试用例都会伤害这些方法的功效。于是就出现了类似于 SAFL 这样为了消除冗余影响的方法。这些方法都用一些实例，或者证明其方法的功效，或者说明该方法怎样解决冗余测试的伤害问题。当然作为一门面向实际的学科，无论怎样强调实验数据都是正确的。不过科学哲学告诉我们，方法只能用理论来证明，实验只是方法的正确性的一种"验证"手段。为此，我们想在一般意义上构造模型来研究上述问题。用测试用例测试程序，其结果应该看成程序语句里透露出来的信息，考虑到熵的概念和信息有关，我们选择信息论理论，构造一个熵模型。这个模型是一个框架，并在一个特殊实例上对 Dicing 方法、TARANTULA 方法、SAFL 方法进行了熵分析，得出了有意义的结果，从理论上基本肯定了文献[3]的结论。

我们今后的工作是为 TBFL 方法建立信息论模型，即认为 X，T 都是随机变量，并为(X，T)的联合分布建模，从而利用 $H(X)$、$H(T)$、$H(X|T)$等工具对 TBFL 方法进行分析。另外冗余测试是否会损害 TBFL 方法的功效，这在直观上是合理的，如同思考线索太多往往影响思路以致找不到问题的关键。但是增加测试在理论上应该有利于问题的解决，所以是否存在这样的信息论方法，当向冗余度较少的测试集里增添新的测试用例时(通常会出现更多的冗余用例)，使得该方法不仅不会对错误定位产生副作用，反而有利于寻找出软件的错误。这也是我们今后研究的另一个目标。

参 考 文 献

[1] Agrawal H, Horgan J, London S, et al. Fault location using execution slices and dataflow tests [C]// IEEE Software Reliability Engineering. 1995: 143-151

[2] Cleve H, Zeller A. Locating causes of program failures [C]// Proceedings of the 27th International Conference on Software Engineering. 2005: 342-351

[3] Hao D, Zhang L, Pan Y, et al. On similarity-awareness in testing-based fault localization [J]. Automated Software Engineering, 2008, 15(2): 207-249

[4] Haykin S. Neural networks-A comprehensive foundation [M]. Beijing: Tsinghua University Press, 2001: 484-508

[5] Kyriazis A, Mathioudakis K. Enhance of fault localization using probabilistic fusion with gas path analysis algorithms [J]. Journal of Engineering for Gas Turbines and Power, 2009, 131(5): 51601-51609

[6] Jones J A, Harrold M J, Stasko J. Visualization of test information to assist fault localization [C]// Proceeding of the 24th International Conference on Software Engineering. 2002: 467-477

[7] Jones J A, Harrold M J. Empirical evaluation of the tarantula automatic fault-localization technique [C]// Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering. 2005: 273-282

[8] Liblit B, Naik M, Zheng A X, et al. Scalable statistical bug isolation [C]// Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI). 2005: 15-16

[9] Schach S R. Object-oriented classical software engineering [M]. Beijing: China Machine Press, 2007: 490-193

[10] Liu C, Yan X, Fei L, et al. SOBER: statistical model-based bug localization [C] // Proceedings of the 13the ACM SIGSOFT Symposium on Foundations of Software Engineering. 2005: 286-295

[11] Renieris M, Reiss S P. Fault localization with nearest neighbor queries [C]// Proceedings of the 18th International Conference on Automated Software Engineering. 2003: 30-39

[12] Zeller A. Isolating cause-effect chains from computer programs [C]// Proceedings of the 10th ACM SIGFOFT Symposium on Foundations of Software Engineering. 2002: 1-10

(上接第 136 页)

[6] Huang R W, Gui X L, Yu S, et al. Study of privacy preserving framework for cloud storage[J]. Computer Scienceand Information Systems, 2011, 8(3): 801-819

[7] Song D, Wagner D, Perrig A. Practical techniques for searcheson encrypted data[C]// Proc. of IEEE Symposium on Security and-Privacy'00. 2000

[8] Boneh D, Crescenzo G D, Ostrovsky R, et al. Public keyencryption with keyword search[C]// Proc. of EUROCRYP'04, volume3027 of LNCS. Springer, 2004

[9] Gentry C. Fully homomorphic encryption using ideal lattices [C]// Proceedings of the 41st ACM Symposiumon Theory of Computing(STOC09). Bethesda, Maryland, USA, 2009: 169-178

[10] Gentry C. Computing arbitrary functions of encrypted data[J]. Communications of the ACM, 2010, 53(3): 97

[11] Swaminathan A, Mao Y, Su G-M, et al. Confidentiality-preserving rank-ordered search[C]// Proc. of the Workshop on Storage Security and Survivability. 2007

[12] Boldyreva A, Chenette N, Lee Y, et al. Order-preservingsymmetric encryption[C]// Proceedings of Eurocrypt'09, volume 5479 of LNCS. Springer, 2009

[13] Wang C, Cao N, Li J, et al. Secure ranked keywordsearch over encrypted cloud data[C]// Proc. of ICDCS'10. 2010

[14] Salton G, Wong A, Yang C S. A Vector Space Model for Automatic Indexing[J]. Communications of the ACM, 1975, 18: 613-620

[15] Zobel J, Moffat A. Inverted files for text search engines [J]. ACM Computing Surverys, 2006, 38(2): 16-31

[16] Rieardo B Y, Berthier R N. ModernInformationRetrieval [M]. NewYork: ACM Press, 1999, 192: 5-10

[17] Salton G, Clement T Y. On The Construction of Effective Vocabularies for Information Retrival[C]// Proceeding of The 1973 Meeting on Programming Languages and Information Retrieval. New York, ACM, 1973: 11