软件学报ISSN 1000-9825, CODEN RUXUEW E-mail: jos@iscas.ac.cn

*Journal of Software*, [doi: 10.13328/j.cnki.jos.000000] http://www.jos.org.cn

©中国科学院软件研究所版权所有. Tel: +86-10-62562563

基于敏感变量和线性感知机的测试预言自动生成方法[[1]](#footnote-1)\*

马春燕1 , 李尚儒1, 王慧朝1, 张磊1, 张涛1

1(西北工业大学 软件与微电子学院, 西安 710072)

通讯作者: 马春燕, E-mail: [machunyan@nwpu.edu.cn](mailto:machunyan@nwpu.edu.cn)

摘 要: 测试预言生成技术是软件工程测试领域的研究热点之一.没有可以利用的历史测试用例是目前大部分测试预言生成技术的普遍假设,但是这种假设会使我们错过利用现有部分测试用例协助自动生成新测试用例预言的机会.在已知部分测试用例集的情况下,提出基于敏感变量和线性感知机的新测试用例的测试预言自动生成方法.首先,收集已知测试用例执行的语句覆盖和不同断点处内存值集合作为训练集,计算与新测试用例执行覆盖信息高度相似的测试用例集;其次,计算各断点处表征成功或失败的敏感变量集;然后,应用线性感知机求解每个断点处成功或失败概率预测的门限值;在此基础上,给出新测试用例测试预言自动生成的方法,并对方法进行讨论.最后,采用7个程序的129个故障版本作为实验对象,对共计14300个测试用例生成测试预言,实证评价表明测试预言准确率平均达到96.2%.论文成果可以形成测试用例集合构造的“滚雪球效应”,不断迭代自动生成新测试用例的测试预言.

关键词: 测试预言;线性感知机;敏感变量;测试用例;内存值

中图法分类号: TP311

中文引用格式: 马春燕,李尚儒,王慧朝,张磊,张涛. 基于线性感知机和敏感变量的测试预言自动生成方法.软件学报. http://www.jos.org.cn/1000-9825/ 0000.htm

英文引用格式: Ma CY, Li SR, Wang HC, Zhang L, Zhang T.. Ruan Jian Xue Bao/Journal of Software, 2018 (in Chinese).http://www.jos.org.cn/1000-9825/0000.htm

Automatic generation method for test oracle based on sensitive variables and linear perceptron

Ma Chun-Yan1, Li Shang-Ru1, Wang Hui-Chao1, Zhang Lei1, Zhang tao1

1(School of software and microelectronics, Northwestern Polytechnical University, xi’an, 710072)

**Abstract**: Test cases generation technology is one of hotspots in the testing field of software engineering. There are no historical test case sets available that are common assumptions about existing test oracle generation techniques. However, this assumption may not always hold, and where it does not, there may be a missed opportunity; perhaps the pre-existing test cases could be used to assist the automated oracle generation of new test cases. In the case of the existing test case set, an automatic test oracle generation method for a new test case based on sensitive variables and linear perceptrons is proposed. First, the statement coverage and memory value set executed by some known test cases are collected, and a set of test cases with high similarity to the execution coverage information of the new test cases is computed. Secondly, the memory sensitive variable set solving algorithm of the program at a given breakpoint are given. Thirdly, the known test case set as the training set and the perceptron is used to solve the threshold value at each breakpoint. And on this base an automatic oracle generation method of the new test case is proposed. Finally, 129 fault versions of seven programs are used as experimental objects to generate test oracles of 14300 new test cases.The empirical evaluation shows that the generated test oracle can achieve 96.2% accuracy on average. The results of the thesis can form the "snowball effect" of the test case set construction, and iteratively automatically generate test oracles for new test cases.

**Key words**: test oracle; linear perceptron; sensitive variables; test cases; memory value

软件测试是保证软件系统质量的主要活动.在软件测试中,测试预言的作用至关重要,它是一种判断程序在给定测试输入下的执行结果是否符合预期的方法.测试预言的质量直接影响测试活动的有效性和软件系统的质量.目前,虽然研究人员已提出了各种自动化测试输入生成技术,成果颇丰,但是测试预言问题仍然被公认为是软件测试中最难解决的问题之一[1].

测试预言问题的综述文献1将测试预言生成的研究分为基于软件形式化规格说明或形式化模型（例如,状态机或时序逻辑规范等）的测试预言生成方法（317篇相关论文）、基于各种软件制品（例如,需求和设计文档、软件执行的属性信息、软件其它版本等）的测试预言生成方法（245篇相关论文）、基于常识或隐含的知识生成测试预言（76篇相关论文）等3类.第1类需要软件开发人员或测试人员给出软件的形式化规格说明,人工参与的工作量较大,形式化规格说明缺乏以及程序行为描述不全面等是该类方法面临的极大挑战.第2类研究克服了第1类研究面临的挑战,正在成为一个活跃的研究方向,主要研究包括基于规格说明或编程接口挖掘的方法、基于程序文档的挖掘方法、蜕变测试、基于源码的静态和动态分析方法,以及基于机器学习的方法等,该类测试预言的生成方法包括全自动的方法或人工辅助的半自动化方法,其中半自动化方法较多. 第3类应用领域较窄,仅针对特种类型或特定的应用范围的软件.

智能化技术在测试预言生成方面的应用研究具有鲜明的学科交叉特点,目前研究成果相对较少,上述第2类测试预言生成方法的研究中逐步开始出现智能化技术的应用.本文工作属于无需测试人员干预的测试预言自动生成方法,借助揭示成功和失败概率的敏感变量,采用监督的机器学习算法-线性感知机作为测试预言自动生成的模型,收集部分测试用例的语句覆盖、不同断点处的内存变量及其取值的集合等数据信息作为训练集,观察新测试用例执行的相应语句覆盖和内存变量集等信息,对该新测试用例在不同断点处的执行结果是成功还是失败进行预测,自动生成测试预言.论文成果可以形成测试用例集合构造的“滚雪球效应”,不断迭代自动生成新测试用例或回归测试用例的测试预言.

本文第1节首先给出相关定义和假设,在此基础上,首先给出求解断点处敏感变量集的算法,并应用线性感知机算法求解断点处成功或故障概率的门限值,提出了本文的测试预言生成算法.第2节对算法进行讨论.第3节给出了实验对象和实验方法.第4节通过案例对算法进行阐释和验证.第5节将已有研究工作进行总结,并与本文工作进行对比分析.

# 测试预言自动生成方法

## 相关定义

本文测试预言生成方法用到的定义如下:

**定义1：测试用例ta和tb的相似度Sim(ta,tb)**

设测试用例ta和tb执行的语句集合分别为A（其元素个数记作|A|）和B（其元素个数记作|B|）,|A∩B|表示A与B交集的元素个数,则ta和tb的相似度定义为：Sim(ta,tb)= 2\*|A∩B|/(|A|+|B|)∈[0,1].Sim(ta,tb)的值越大,ta和tb的相似度越高,如果A=B,则Sim(ta,tb)=1,ta和tb的相似度最高;如果A∩B=Ф,则Sim(ta,tb)=0,ta和tb的相似度最低.

**定义2：内存变量**

内存变量是一个二元式[var,val],其中,var表示变量名,val表示该变量的值.

**定义3：两个内存变量相等**

两个内存变量[vara,vala]和[varb,valb]相等当且仅当vara=varb和vala=valb

**定义4：两个内存变量集合M和N的交集**

两个内存变量集合M和N的交集记作M∩N,定义为M与N中相等的内存变量的集合

**定义5：两个内存变量集合*M*和*N*的差集**

两个内存变量集合***M***和***N***的差集记作***M-N***,定义为与***N***中变量名相同而变量值不同的M中内存变量的集合

**定义6：敏感变量的定义**

一个敏感变量***t***是一个内存变量二元式***t***=[***sv***, ***v***],其中,***sv***表示敏感变量的名字,***v***表示敏感变量的值

## 相关假设:

根据程序测试和调试经验,本文的算法思想基于下述基本假设：

**假设1:变量集对成功和失败具有一定的揭示能力**

对于高相似度的测试用例(见定义1)形成的一个集合***St***而言,在每个断点处,***St***中测试用例执行的变量集合的交集或差集对程序执行的成功概率或失败概率具有一定的揭示能力.

**假设2:不同断点处变量集合对成功或故障的揭示能力不同**

因为故障发生程序的某个位置且故障有传播能力,断点位置越靠后,其相应变量集合对成功或故障的揭示能力逐步增强.

## 测试预言生成算法

已知一个程序、一个测试用例集合及各测试用例执行成功或失败的信息,本文通过在程序中设置批量断点的方法,收集所有测试用例在各断点处的内存数据以及代码覆盖信息作为训练集,然后,根据新测试用例的执行,计算各断点处的敏感变量集合,并应用线性感知机算法计算该断点处执行失败或成功概率的门限值,在此基础上,本节给出了一种基于内存分析的测试预言自动生成方法,即对新产生的测试用例执行同一程序的运行结果是成功还是失败进行预测,自动生成测试预言(见下文算法1).为了阐述本文提出的测试预言生成算法,表1给出了算法中涉及的符号名称及其含义描述.

表**1** 本节算法中涉及的符号名称及其含义描述

|  |  |  |
| --- | --- | --- |
| 符号名称 | 英文全称 | 含义描述 |
| ***P*** | Program | 故障程序 |
| ***ST*** | **S**et of *n* **T**est cass | ***n***个测试用例的集合***{[t1,o1],[ t2,o2],[ t3,o3],……,[ tn,on]}***  ***ti***是第***i***个测试输入,***oi***是第***i***个测试输入的执行结果(成功或失败), ***oi***为0表示***ti***执行失败,***oi***为1表示***ti***执行成功,在算法描述中也用***Ti***表示第***i***个测试用例 |
| ***SB*** | **S**et of *m***B**reakpoints | 程序***P***中设置的***m***个断点集合***{b1,b2,b3,…,bm}***,***bi***表示第***i***个断点 |
| ***SSB*** | **S**et of all **S**ensitive variables for *m* **B**reakpoints | 程序***P***中***m***个断点处各敏感变量集合构成的集合***{SBb1, SBb2, SBb3,…, SBbm}***, ***SBbi*表示第*i*个断点*bi*处的敏感变量集合** |
| ***tnew*** |  | 表示一个待生成测试预言的新测试用例***[t,o],t***是测试输入,***o***为0表示***t***执行失败,***o***为1表示***t***执行成功 |
| ***PPi*** | **P**ropobility of **P**assing for *tnew* under breakpoint *bi* | ***Pi***表示***tnew***第在***i(1≤i≤m)***个断点***bi***处执行成功的概率 |
| ***PP*** | **P**ropobility of **P**assing for *tnew* | 表示***tnew***执行成功的概率 |
| ***STtnew*** | **S**et of **T**est cases that are most similar to ***tnew*** | 表示***ST***中与***tnew***最相似的测试用例集 |
| ***SPTtnew*** | **S**et of **P**assing test cases in ***STtnew*** | 表示***STtnew***中成功测试用例组成的集合 |
| ***SFTtnew*** | **S**et of **F**ailing test cases in ***STtnew*** | 表示***STtnew***中失败测试用例组成的集合 |

### 1.3.1 算法1测试预言生成算法

给定***P***, ***ST***, ***SB***和***tnew***的测试输入***t***,首先计算***ST***中与***tnew***最相似的测试用例集合***STtnew***.（见下述算法1为伪代码中的1-20行）.将获得的***STtnew***,分为最相似成功测试用例集***SPTtnew***,和失败测试用例集***SFTtnew***,调用**算法2**计算***SB***中每个断点处的敏感变量集.

* 如果***SPTtnew≠*Ф*, SFTtnew≠*Ф*,***求得的敏感变量的含义为：从所有与***tnew***相似正确测试用例和失败测试用例的执行来看,可能错误的内存变量有哪些,在这种情况下,敏感变量称为“故障敏感变量”
* 如果***SPTtnew≠*Ф*, SFTtnew =*Ф*,***求得的敏感变量的含义为：从所有与***tnew***相似的正确测试用例的执行来看,可能正确的内存变量有哪些,在这种情况下,敏感变量称为“成功敏感变量”
* 如果***SPTtnew =*Ф*, SFTtnew≠*Ф*,***求得的敏感变量的含义为：从所有与***tnew***相似的失败测试用例的执行来看,可能错误的内存变量有哪些,在这种情况下,敏感变量称为“故障敏感变量”

然后调用算法3,计算***bi***处的门限值***Thresholdbi***,根据算法1的21-79行陆续计算各断点处程序执行成功的概率***PP1***,***PP2***,***PP3***,…,***PPm***.

最后对于序列***PP1***, ***PP2***, ***PP3***,…, ***PPm*,** 用等差序列作为对预测结果影响所占的权重,分别记为***a1***, ***a2***, ***a3***, … ***am***,则等差数列的公差为***d***=***(am–a1)/(m-1***),可以求得该等差数列首项***a1***=***1/m-(am–a1)/2***=***3/4m***,其余各项为***ai***=***a1+(i-1)d***,测试用例执行程序***P***成功的概率为***PP***=***PP1 \* a1***+ ***PP2 \* a2***…+ ***PPm \* am***.

**算法1的伪代码如下所示（1-88行）：**

Algorithm:GeneratingTestOracles

Input: *P*, *ST*, *SB*和the test input of *tnew*.

Output：Test oracle of *tnew.*

1. *temp←tnew*
2. *STtnew←Ф*
3. *While(STtnew!=Ф)*
4. *C←Ф*
5. MaxSimularTestCases*←Ф*
6. Largenumber*←0*
7. for(each *ti* in *ST*)
8. if *Sim(temp,ti)=1*
9. *C← C∪{ ti}*
10. *STtnew← STtnew∪C*
11. endfor
12. if(*C*=*Ф*)
13. for(each *ti* in *ST*)
14. if (*Sim(temp,ti)>largenum*)
15. largenum*← Sim(temp,ti)*
16. temp*← ti*
17. Endif
18. endfor
19. endif
20. endwile
21. *SPTtnew← Select the set of the passing test cases from STtnew*
22. *SFTtnew← Select the set of the failing test cases from STtnew*
23. for(each *bi* in *SB*)
24. *SBbi* *←*CalculateSensitiveVariabls（*P, SPTtnew, SFTtnew, bi*）
25. Endfor
26. *m←0*
27. for(each *bi* in *SB*)
28. *Thresholdbi←Threshold(P, SPTtnew, SFTtnew, bi, SBbi)*
29. *FCi←0*
30. *PCi←0*
31. *if(SPTtnew*!=Ф, *SFTtnew*!=Ф*)*
32. for(each test case t*j* in *SPTtnew*)
33. *Mt←* *get memory variables set from the execution of tnew forbi*
34. *Mtj←* *get memory variables set from the execution of tj forbi*
35. *XCj← (Mt-Mtj)∩SBbi*
36. If(|*XCj*|/|(*Mt-M tj*)|>*Thresholdbi*)
37. *FCi*=*FCi+1*
38. else
39. *PCi*=*PCi*+*1*
40. endif
41. endfor
42. for(each test case t*j* in *SFTtnew*)
43. *Mt←* *get memory variables set from the execution of tnew forbi*
44. *Mtj←* *get memory variables set from the execution of tj forbi*
45. *XCj← Mt∩Mtj∩SBbi*
46. If(|*XCj*|/|(*Mt-M tj*)|>*Thresholdbi*)
47. *FCi*=*FCi+1*
48. Else
49. *PCi*=*PCi*+*1*
50. endif
51. endfor
52. else if(*SPTtnew*!=Ф, *SFTtnew*=Ф)
53. for(each test case t*j* in *SPTtnew*)
54. *Mt←* *get memory variables set from the execution of tnew forbi*
55. *Mtj←* *get memory variables set from the execution of tj forbi*
56. *XCj← Mt∩Mtj∩SBbi*
57. if(|*XCj*|/|(*Mt-M tj*)|<*Thresholdbi*)
58. *FCi*=*FCi+1*
59. else
60. *PCi*=*PCi*+*1*
61. endif
62. endfor
63. else if(*SPTtnew*=Ф, *SFTtnew*!=Ф)
64. for(each test case t*j* in *SFTtnew*)
65. *Mt←* *get memory variables set from the execution of tnew forbi*
66. *Mtj←* *get memory variables set from the execution of tj forbi*
67. *XCj← Mt∩Mtj∩SBbi*
68. If(|*XCj*|/|(*Mt-M tj*)|>*Thresholdbi*)
69. *FCi*=*FCi+1*
70. else
71. *PCi*=*PCi*+*1*
72. endif
73. endfor
74. endif
75. *PPi←PCi/(PCi+FCi)*
76. m*←m+1*
77. Endfor
78. *d← ( am–a1)/(m-1*)
79. *a1←1/m-(am–a1)/2*=*3/4m*
80. *ai←a1+(i-1)d*
81. *PP←PP1 \* a1*+ *PP2 \* a2*…+ *PPm \* am*
82. if *PP>0.5*
83. *return 1*
84. else
85. *return 0*
86. endif

### 1.3.2 算法2敏感变量集求解算法

给定输入*P, SPTtnew*, *SFTtnew*, 以及*SB*中第*i*个断点*bi*. 设两个临时变量*ST1和ST2*,

* 如果*SPTtnew≠Ф*, *SFTtnew≠Ф*,则令*ST1= SPTtnew*, *ST2= SFTtnew*
* 如果*SPTtnew≠Ф*, *SFTtnew =Ф*,则令*ST1= SPTtnew*, *ST2= SPTtnew*
* 如果*SPTtnew =Ф*, *SFTtnew≠Ф*,则令*ST1= SFTtnew*, *ST2= SFTtnew*

令x=| *ST1*|, y=|*ST2*|,首先计算得到个内存变量集合{*dj1*, *dj2*, *dj3*,……,*djy*},将其存入哈希表*Dpj*,其中*djk*（*1≤k≤y*）作为哈希表的*key*（记作*djk.key*）,该内存变量出现的次数作为哈希表的*value*（记作*djk.value*）

然后,可计算得到的*x*个哈希表*Dp1*, *Dp2*, *Dp3*,…, *Dpx*,计算哈希表*Hbi*=*Dp1*∪*Dp2*∪…∪*DpX*,其中∪表示的含义为：各个*Dpj*（*1≤j≤| SFTtnew |*）中*key*不同则合并,*key*相同则*value*相加.

最后对*Hbi* 中各*value*值从大到小排序,*key值*为前20%的变量(取key值较大的变量作为敏感变量)加入*SBbi*,算法结束,返回*SBbi*.

算法2的伪代码如下所示(1-29行)：

Algorithm:CalculateSensitiveVariabls

Input: *P, SPTtnew*, *SFTtnew*, and the *i*-th breakpoint *bi* in *SB*.

Output：*SBbi* which is the set of sensitive variables for the *i*-th breakpoint *bi* in *SB.*

1. if(*SPTtnew!= Ф*&&*SFTtnew!=Ф*)
2. *ST1← SPTtnew*
3. *ST2← SFTtnew*
4. else if(*SPTtnew!=Ф*, *SFTtnew =Ф*)
5. *ST1← SPTtnew*
6. *ST2← SPTtnew*
7. else if(*SPTtnew =Ф*, *SFTtnew!=Ф*)
8. *ST1← SFTtnew*
9. *ST2← SFTtnew*
10. end if
11. j*←1*
12. for(each test case *t1j*,from *ST1*)
13. k*←1*
14. for(each test case *t2k*,from *ST2*)
15. M *←get memory variables set from the execution of t1j*
16. N *←get memory variables set from the execution of t2k*
17. if(*ST1!= ST2*)
18. *djk←M-N*
19. else if
20. *djk←M*∩*N*
21. endif
22. hashMap *Dpj(key,value)←( djk, the cumulative number of timesof djk appearing)*
23. k++
24. endfor
25. j++
26. endfor
27. *Hbi**←Dp1*∪*Dp2*∪…∪*DpX*,different keys are merged and the value is summed for the same key
28. *SBbi ←selected the top 20% from the* sorted *Hbi* according to the value in the hasmap
29. return *SBbi.*

### 1.3.2 算法3 应用线性感知机算法求解每个断点处门限值

给定输入*P, SPTtnew*, *SFTtnew*, *bi*以及*SBbi*, 首先初始化候选门限值二维数组*temp[][]*为零,两维数组长度都为*SPTtnew∪SFTtnew*中测试用例个数.对于*SPTtnew∪SFTtnew*中每个测试用例*tk*(*1≤k≤| SPTtnew∪SFTtnew |*),将其作为一个新测试用例, 分别计算*temp*的第*k*行元素,根据“测试预言生成算法”,对于*SPTtnew∪SFTtnew*中每个测试用例*tk*(*1≤k≤| SPTtnew∪SFTtnew |*),将其作为一个新测试用例*tnew*；计算每个候选门限值的预测成功率,将*temp*中预测准确率最大的候选门限值存入*Thresholdbi* ,算法结束,返回*Thresholdbi*.

算法3的伪代码如下所示（1-34行）：

Algorithm: *Threshold*

Input: *P, SPTtnew, SFTtnew, bi, SBbi*

Output：*Thresholdbi for bi.*

1. max←|*SPTtnew∪SFTtnew*|
2. *temp[max][max]*←*0*
3. *for(* each test case t*k* in *STtnew*)
4. *if(SPTtnew*!=Ф, *SFTtnew*!=Ф*)*
5. for(each test case t*j* in *SPTtnew*)
6. *Mt←* *get memory variables set from the execution of tnew forbi*
7. *Mtj←* *get memory variables set from the execution of tj forbi*
8. *XCj← (Mt-Mtj)∩SBbi*
9. *temp[k][j]*←|*XCj*|/|(*Mt-M tj*)|
10. endfor
11. for(each test case t*j* in *SFTtnew*)
12. *Mt←* *get memory variables set from the execution of tnew forbi*
13. *Mtj←* *get memory variables set from the execution of tj forbi*
14. *XCj← Mt∩Mtj∩SBbi*
15. *temp[k][j]*←|*XCj*|/|(*Mt∩M tj*)|
16. endfor
17. else if(*SPTtnew*!=Ф, *SFTtnew*=Ф)
18. for(each test case t*j* in *SPTtnew*)
19. *Mt←* *get memory variables set from the execution of tnew forbi*
20. *Mtj←* *get memory variables set from the execution of tj forbi*
21. *XCj← Mt∩Mtj∩SBbi*
22. *temp[k][j]*←|*XCj*|/|(*Mt∩M tj*)|
23. Endfor
24. else if(*SPTtnew*=Ф, *SFTtnew*!=Ф)
25. for(each test case t*j* in *SFTtnew*)
26. *Mt←* *get memory variables set from the execution of tnew forbi*
27. *Mtj←* *get memory variables set from the execution of tj forbi*
28. *XCj← Mt∩Mtj∩SBbi*
29. *temp[k][j]*←|*XCj*|/|(*Mt∩M tj*)|
30. endfor
31. endif
32. endfor
33. *Thresholdbi*←select the value that has the highest prediction accuracy in training set.
34. *return Thresholdbi*

# 实验对象及实验方法

论文采用SIR（Software-artifact Infrastructure Repository）知识库[18]提供的5个C程序实验对象和从开源网站travis-ci上下载的两个C程序实验对象expresstionParser和sort (网址：https://travis-ci.org/swenson/),通过实际案例程序阐释本文给出的测试预言自动生成方法的有效性.表1给出了实例验证过程中所使用的每个案例名、功能、故障版本数、测试用例数以及正确版本的源码行数等信息.

**Table 1** Experimental objects

表**1** 实验对象

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 程序名 | 功能描述 | 故障版本数 | 测试用例数 | 代码行数 |
| tcas | 空中防撞系统 | 41 | 1500 | 174 |
| print\_tokens | 词法分析器 | 7 | 4130 | 726 |
| schedule2 | 优先级调度器 | 10 | 2710 | 374 |
| replace | 模式替换 | 32 | 5542 | 564 |
| tot\_info | 信息统计 | 23 | 1052 | 565 |
| expressionParser | 表达式解析和运算 | 10 | 1500 | 1251 |
| sort | 各类排序算法比较 | 10 | 1500 | 2512 |

为将提出的测试预言生成算法进行实践,论文开发了图1所示的实验辅助工具,工具的开发环境是Ubuntu Linux 10.04.为了方便与SQLite数据库以及GDB调试集成,论文应用Python语言和shell script实现工具的开发,辅助工具源码TestOracleGen.rar在<https://github.com/machunyan/LCEC>网站上可以公开下载.工具的主要模块功能阐述如下：

1. **程序代码分析模块.**该模块负责分析源文件,根据断点的设置原则进行断点标注（本文实验过程中设置的断点设置原则为：以return和exit作为结束的行,本实验辅助工具允许用户批量自定义断点位置）,为程序执行以及执行过程中调用GDB命令获取内存数据做准备.
2. **程序执行分析模块.**该模块运行每一个测试用例,在程序运行过程中获取断点处程序执行的内存变量集,并通过通过gcov软件,获取每个测试用例运行时的程序覆盖信息.
3. **数据存储模块.**该模块将程序代码分析模块和程序执行分析模块得到的断点信息和测试用例的执行覆盖信息存入数据库.在分析所有测试用例执行覆盖的基础上,对不同覆盖进行分类,计算测试用例之间执行覆盖的联系.
4. **预言模块.**该模块根据上述三个模块收集的故障程序的断点信息、测试用例运行故障程序时所有断点的内存变量信息,以及执行语句覆盖信息,执行1.2节中的测试预言生成算法,对测试用例运行同一故障程序的运行结果是成功还是失败进行预测.

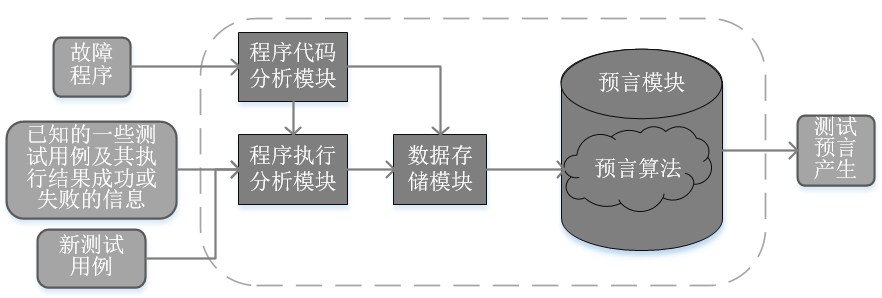


Fig.1 The related assistant tool framework for experiments

图1 实验工具总体架构

采用该实验辅助工具的具体实验步骤如下：

1. 将工具复制到需要进行测试预言生成的测试程序所在的文件夹中.
2. 在测试程序的文件夹中建立output文件夹,用于存放程序的执行覆盖文件cov\_result和mod\_classify\_result.
3. 将已知执行结果的所有测试用例存放在文件trainset中.
4. 运行数据收集程序train.sh,输入参数为已知结果的测试用例存放的文件名,测试程序对象名,测试故障程序版本号.数据收集程序执行完成后,将收集的所有数据信息存放于数据库中.
5. 运行预测程序predict.sh,计算新测试用例的测试预言.

# 实验结果及分析

## 实验对象的实验结果及分析

Tcas、print\_tokens、schedule2、replace、tot\_info、expressionParser和sort共129个故障版本.为每一个故障版本设计100个新的测试用例（其中成功和失败的比率与训练集一致）,计算这100个测试用例预言正确的个数.图2~8分别展示了每类程序所有故障版本的预言正确数目的折线图.图中横轴表示每类程序的版本号,竖轴表示100个新测试用例使用本文的测试预言自动生成方法预测正确的个数.

程序tcas共有41个故障版本,每个版本运行100个新的测试用例,预测情况如图5-1所示.预测效果最差的是V39,91个测试用例预测正确.V3、V7等8个故障版本的100个测试用例全部预测正确.41个tcas版本预言生成的正确率平均为96.3%.

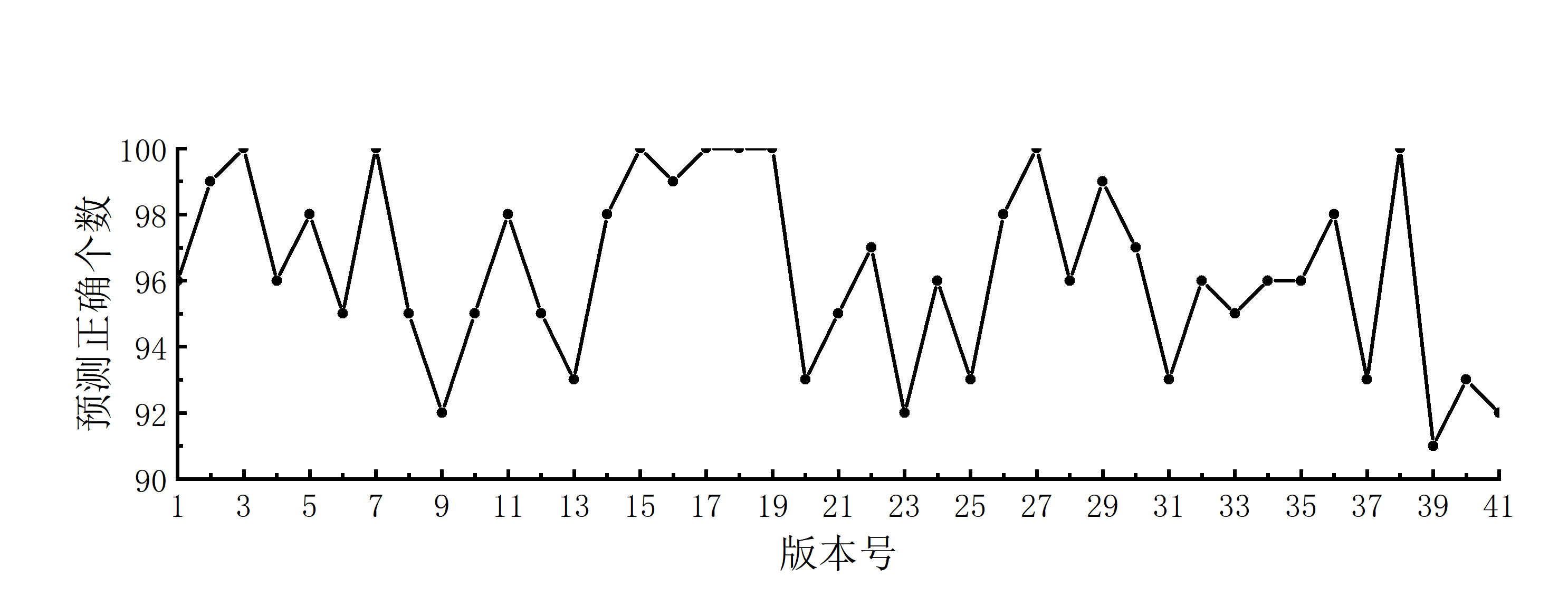


Fig.2 Test oracle prediction results of 41 tcas versions

图2 tcas的41个版本的预测结果

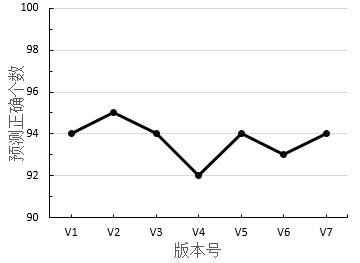
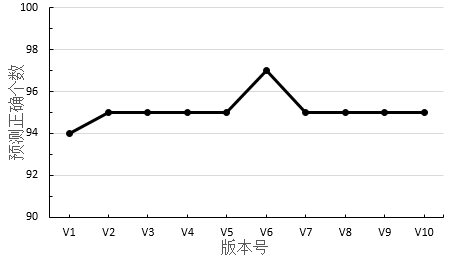
 

Fig.3 Test oracle prediction results of 7 print\_tokens versions Fig.4 Test oracle prediction results of 10

图3 print\_tokens的7个版本的预测结果 schedule2 version

图4 schedule2的10个版本的预测结果

程序print\_tokens共有7个故障版本,每个版本运行100个新的测试用例,预测情况如图3所示.预测效果最差的是版本V4, 92个测试用例预测正确.预测正确个数最多的是版本V2,预测正确95个.7个版本平均预言正确率为93.7%.

程序schedule2共有10个故障版本,每个版本运行100个新的测试用例,预测情况如图4所示.故障程序schedule2的预测效果比较平均,有八个版本预测正确95个.10个版本平均预言正确率为95.1%.

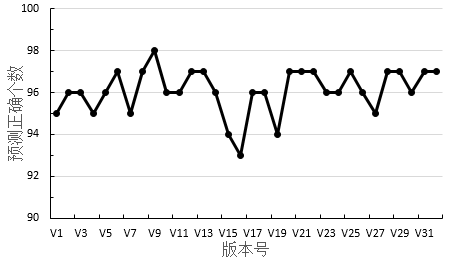
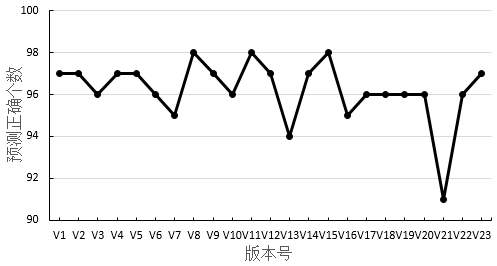
 

Fig.5 Test oracle prediction results of 32 replace versions Fig.6 Test oracle prediction results of 23

图5 replace的32个版本的预测结果 tot\_info versions

图6 tot\_info的23个版本的预测结果

程序replace共有32个故障版本,每个版本运行100个新的测试用例,预测情况如图5所示.预测效果最差的是版本V16,93个测试用例预测正确.预测正确个数最多的是版本V9,预测正确98个.32个版本平均预言正确率为96.1%.

程序tot\_info共有23个故障版本,每个版本运行100个新的测试用例,预测情况如图6所示.预测效果最差的是版本V21, 91个测试用例预测正确.预测正确个数最多的有版本V8、V11和V15,预测正确98个.23个版本平均预言正确率为96.2%.

程序expressionParser、sort各有8个故障版本,每个版本运行100个新的测试用例,预测情况如图7所示.8个版本平均预言正确率分别为97.5%和98.6%.

综上,对129个故障版本,每个版本预言100个测试用例,平均预言正确率为96.2%,说明了本文给出的测试预言自动生成方法预测的有效性.

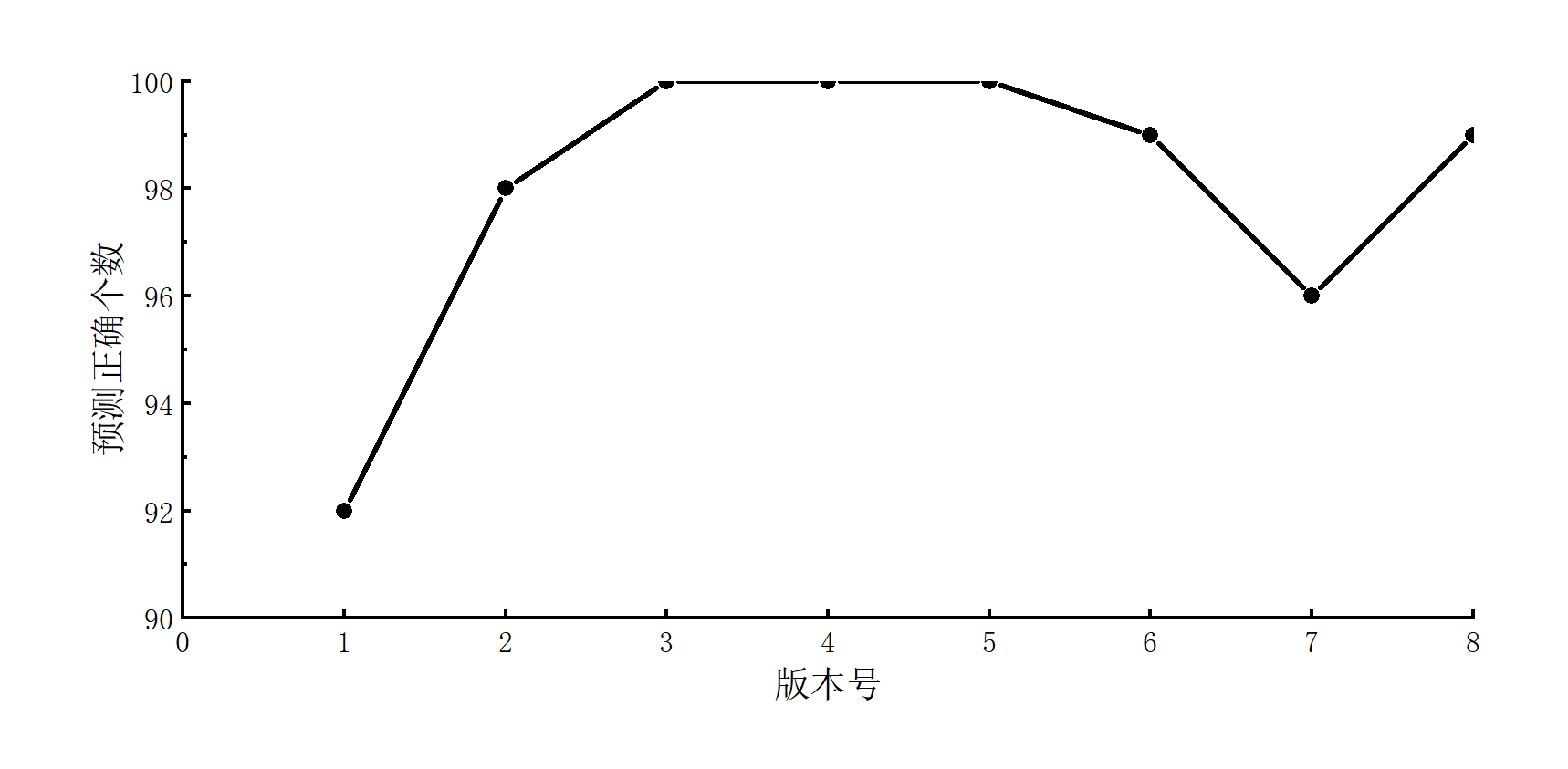
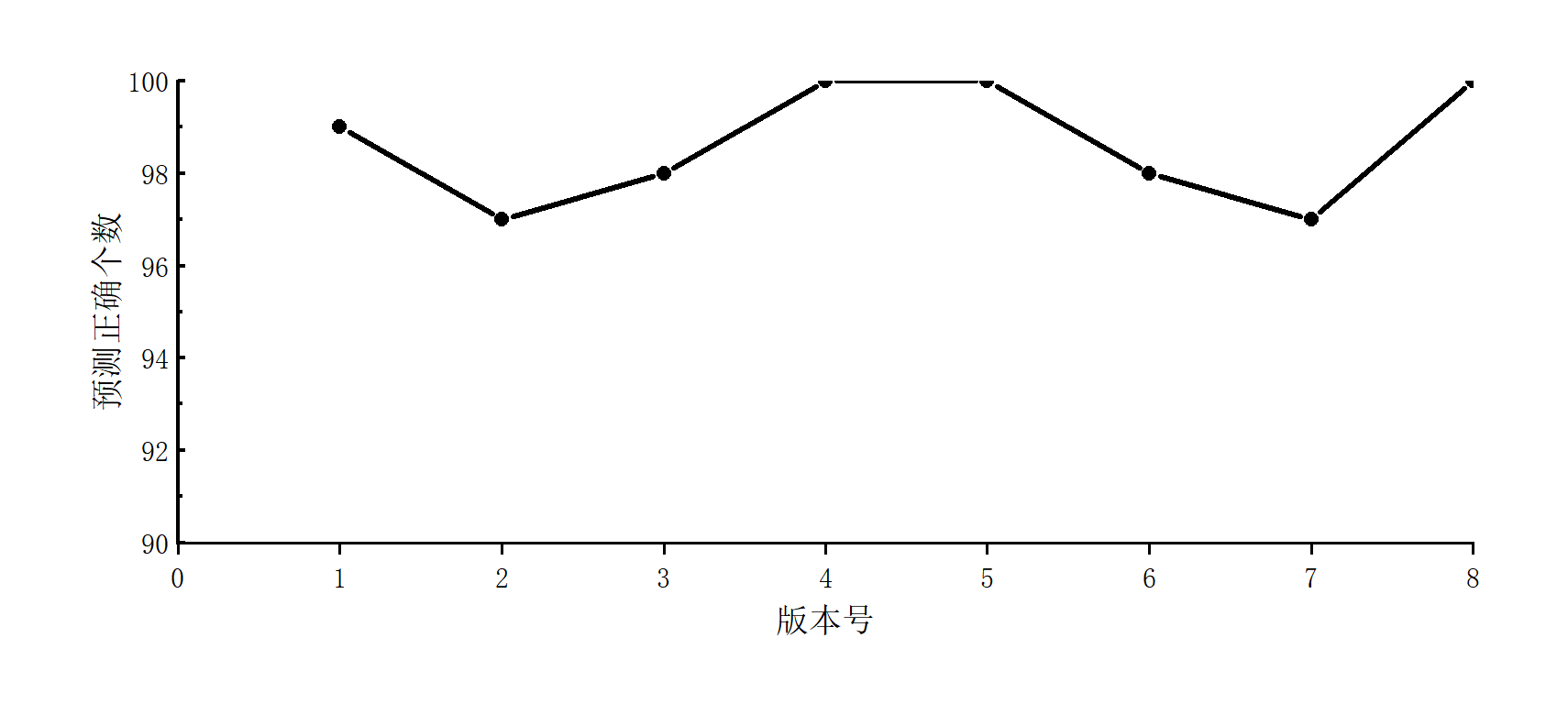
 

Fig.7 Test oracle prediction results of 8 expressionParser versions Fig. 8 Test oracle prediction results of 8 sort

图7 expressionParser 8个版本的预测结 图8 sort 8个版本的预测结果

## 实验结果与其它相似方法的比较

从采用机器学习、现有测试用例的输入及执行结果协助自动生成新测试预言和完全自动化的角度来看,与本文相似的工作有文献13、14和17等.与他们相比本文工作的优势如下:

1. 本文训练集数据量要求宽松,文献17要求失败的输入和成功的输入各占一半,而且数据量偏大,在实际应用中不可行.
2. 本文测试预言的生成应用范围更广.针对某个新的测试输入,上述3个文献生成的测试预言准确率完全依赖于训练集中的测试输入特征,在具体应用中具有一定的局限性和针对性;而本文的方法则是收集训练集中测试输入执行的语句覆盖、不同断点处的内存变量及其取值的集合等大量数据信息,更能真实反映被测试程序的行为,对于哪些输入数据个数少,输入取值有限或程序执行是否成功与输入的对应关系不明显等的应用程序而言,本文的的方法皆可应用.
3. 本文的实证研究更深入和合理.上述文献的实证研究均通过个别案例程序并注入若干故障来检测方法的有效性.从故障版本的数量(共113版本)、故障类型的数量(比较和逻辑等各类运算符错误、常量赋值错误、变量定义错误、语句丢失、多余语句、变量赋值错误)和产生的测试预言数量(11300=113\*100)角度,本文的实验更加深入和合理.
4. 测试预言的准确率方面本文有一定优势.从文献17的实验结果来看,对于案例Tcas而言,Sir知识库中有效测试用例共1500个,其中成功测试用例约占85%,作者在现有测试用例的基础上又构造了1500个测试用例作为训练集,其预测准确率为98.72%,但在1500个测试数据集(还要求成功输入与失败输入数量相当)的情况下,其预测其预测准确率约为82%(作者未指明是否针对Sir知识库中41个版本都进行了实验,也未指明测试预言产生的数量),而本文针对Tcas的41个版本,每个版本均生成了100个测试输入的测试预言,平均预测准确率为96.3%,其中Tcas的第3,7,15,17-20和28等8个版本的预测准确率为100%.文献17中的其他实验对象,预测准确率最低为84.66%.本文在其它实验对象中,在最少训练集合1052个的情况下,预测准确率最低平均为95.1%.

# 方法讨论

## 方法的实用性

本文提出的测试预言自动生成方法独立于源代码,不需要向源代码中插入任何断言,独立于编程语言,对任何一种编程语言都适用,没有构造形式模型的限制,具有很强的灵活性,适用范围广.本文的测试预言预测方法也适用于其它编程语言,例如,Java和C++语言的程序,但是本文搭建的实验环境仅支持C语言程序.

在软件调试过程中,程序员需要通过在程序关键语句处加断点,通过分析和比较部分断点处的变量取值以及语句执行序列等上下文来诊断故障,这是故障诊断的核心行为.所以本文假设符合实际情况,这一点在文献[19]的研究成果中也得到印证.

算法1生成的***tnew***测试预言的有效性与集合***T***和***St***密切相关,如果集合***T***或***St的元素个数非常少,***则算法输出结果的准确率就会降低.该算法建立在已经存在的测试用例集合***T之***上,要求集合***T***存在一定数量,根据目前程序（C,C++,Java等语言程序）的特点,仅有分支或while循环的判断条件可能导致测试用例执行的路径不同,所以,在实际求解中,***T***中与***tnew***最相似的测试用例集***St***也会满足算法求解的要求.

算法1的有效性并不依赖于***T***中测试用例的类型,并不要求***T***必须包含成功测试用例或必须包含失败测试用例.***T***可以仅包含成功测试用例,也可以仅包含失败测试用例,也可以既包含成功测试用例又包含失败测试用例.所以,本文训练集数据量要求宽松,并不要求失败的输入和成功的输入各占一半保持均衡.

本文测试预言生成算法在实际项目应用中可以迭代“滚雪球”式生成新的高质量测试用例.例如,在项目真实的测试实践中,如果已经运行了100个测试用例,都执行成功,那么对若干新的测试输入,可以用测试预言生成算法预测他们的输出成功还是失败,提示用户重点关注预测失败的新测试输入的执行.

## 测试预言生成的准确率问题

本文生成的某个新测试用例*tnew*的测试预言准确率与下述几个方面的要素密切相关:

1. 给定***tnew***,与***tnew***相似测试用例集合***St***相关***, St***元素个数越多, 训练集质量越高,预测结果趋于越精确.
2. 与各断点处敏感变量集的成功或失败的揭示能力相关,例如,枚举变量、布尔变量及取值为离散值的变量,测试执行时,它们的取值较为集中而且重复率较高,所以,这些变量一定被作为敏感变量求解出来,他们揭示程序成功或失败的能力更强.

本文通过分析不同版本的故障发现,如果被测试程序的故障语句是每个测试用例一定执行的语句,与***tnew***相似测试用例个数较多,我们的方法生成测试预言的准确率为100%,例如Tcas的V3、V7等V8版本.如果被测试程序的故障语句是if或while中的语句,其对于测试输入集合而言,故障语句未必遇到每个测试用例都执行,故障语句执行次数越少,预测准确率就越低.主要原因是故障语句执行次数少,导致与预测测试用例高度相似的测试用例过少,并且故障敏感变量的影响指标(揭示成功或失败的能力)明显下降,例如,print\_tokens的故障版本v4,tcas的故障版本v33,和replace的故障版本v16等,它们的故障语句都在多层嵌套的if判断条件中,通过其它条件不变,增加相似测试用例的控制实验,我们发现如果增加相似测试用例集合中的个数,它们的预测结果会进一步提高.这也印证了与测试预言准确率相关的上述第1)和2）要素.

## 方法成本的评估

目前,本文的方法未考虑多线程程序,以及通过界面与用户交互的图形界面程序.测试套件下被测试程序各断点处变量值集合相关信息收集的工作量较大,它是测试预言生成算法的主要开销,也是本文的主要缺点和代价.

测试预言预测速度受断点个数P、全部测试用例用例的个数T以及与待预测测试用例相似的测试用例个数N的影响.算法1计算相似测试用例集合的复杂度为O(T2),每增加一个断,算法1就要在该断点处执行获取内存数据、计算敏感变量集和门限值等操作,因此算法1的效率与P呈线性关系.算法2比较相似测试用例集合中成功输入（令程序执行成功的输入）与失败输入（令程序执行失败的输入）的内存变量与其取值,计算敏感变量集合,该计算过程的复杂度为O(N2) .最后,算法3在每个断点处计算门限值,所以每次计算的复杂度是O(N2).由于算法1调用了算法2和算法3,综上所述,算法1的算法复杂度较大,为O(T2+P×N2).

# 相关工作及结论

一些研究成果需要测试人员手动输入规格说明,或模型,或训练神经网络的方法,辅助部分测试预言的自动生成.例如,1)在回归测试中,一些研究基于规格说明和组件应用编程接口规范,应用人工神经网络自动生成测试预言[3-6],这些方法需要手工构造规格说明; 2)由测试者首先定义包含蜕变关系和数据挖掘算法的JML规格说明,然后采用[蜕变测试技术](http://www.baidu.com/link?url=F3awWil8zXoSY3jeWimUEm-AA-1E7tMFx4S2w55sBuHjlGuVhkn7GTvPkVCi7B9kbsiSZvhOxkqNcAaYuO1zta)来产生测试预言[7],中文参考文献[1]通过实验研究蜕变测试技术产生测试预言的有效性; 3)ASTOOT通过检查两个不同的执行场景来产生测试预言,但是需要测试人员提供被测试系统的代数规格说明[8]; 4)Li N提出了基于模型的测试预言生成策略,包括采用状态不变量作为基于状态图测试预言生成的依据[9]; 5) Guo H F. A等首次提出一个通用的声明框架,该框架基于测试人员撰写的语义和控制流图生成测试预言[12]; 6)Wang等[15]利用支持向量机SVM作为监督的机器学习算法,测试人员采用智能测试预言库对被测试程序代码进行注释,根据函数调用收集测试跟踪信息,对训练集中每个测试轨迹提取特征作为SVM算法的输入,然后使用构建的SVM模型作为测试预言,作者通过两个案例程序的实验结果阐释了方法的有效性.

无需测试人员干预的测试预言自动生成方法.Arantes等采用逆向工程,通过静态和动态源代码分析的方式构造程序的一个结构化模型,然后基于该模型自动生成测试预言[10],提出的方法检测4了个案例中的部分故障.Goffi A等借助现代软件系统可以通过不同的执行顺序提供相同的功能的特征,通过检查给定执行序列与所有可用的冗余和假设等效的执行序列之间的等价性,设计和开发了一种完全自动化产生测试预言的技术,方法生成的测试预言发现了多达53％的错误,并对开发人员撰写的16个案例中4个案例的测试预言,应用该方法提升了其有效性[11].Shahamiri et al.以被测试程序输入空间的行为作为训练集合,应用神经网络生成新测试集输入的测试预言,然后该测试预言的预期结果与被测试程序真实的输出进行比对[13-14],最后,作者通过两个案例阐释其研究成果的有效性,由于被测试单元没有输出或输出形式的多样性,该方法以被测试单元的输出域作为准则产生测试预言有很大的局限性.Farshad Gholami[17]等以测试输入及其对应的成功或失败标记作为训练集,应用神经网络算法建立一个模型,为测试输入产生测试预言,作者通过两个案例研究阐释了其方法的有效性,但是该方法需要的训练集数量较大,而且要求失败的输入和成功的输入各占一半,而现实中积累大量执行失败的输入是不可能的.

本文工作属于无需测试人员干预的测试预言自动生成方法.本文采用监督的机器学习算法线性感知机作为测试预言自动生成的模型, 首先,论文收集部分已知测试用例执行的语句覆盖和内存值集合,给出了程序在给定断点处的内存中敏感变量集求解算法;然后,将已知测试用例集合作为训练集,应用线性感知机求解每个断点处的门限值;最后提出新测试用例的测试预言自动生成方法.

未来工作将采用大型真实案例对本文提出的测试预言自动生成方法进行进一步分析与验证,同时,进一步探索测试预测的准确率与故障类型的关系.

References:

1. Barr E T, Harman M, Mcminn P, et al. The Oracle Problem in Software Testing: A Survey[J]. IEEE Transactions on Software Engineering, 2015, 41(5):507-525.
2. E. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, “Repository of publications on the test oracle problem,” 2015. [Online]. Available: http://crestweb.cs.ucl.ac.uk/resources/ oracle\_repository/, Accessed on: Jul. 2015.
3. Memon A M, Pollack M E, Soffa M L. Automated test oracles for GUIs[C]// Century Applications. ACM, 2000:30-39.
4. Ye M, Feng B, Zhu L, et al. Automated Test Oracle Based on Neural Networks[C]// IEEE International Conference on Cognitive Informatics. IEEE, 2007:517-522.
5. Jin H, Wang Y, Chen N W, et al. Artificial Neural Network for Automatic Test Oracles Generation[C]// International Conference on Computer Science and Software Engineering. IEEE Computer Society, 2008:727-730.
6. Xu W, Ding T, Wang H, et al. Mining Test Oracles for Test Inputs Generated from Java Bytecode[C]// Computer Software and Applications Conference. IEEE, 2013:27-32.
7. C. Murphy, K. Shen and G. Kaiser, Using JML Runtime Assertion Checking to Automate Metamorphic Testing in Applications without Test Oracles, Proceedings of the 2009 International Conference on Software Testing Verification and Validation, ICST ’09, IEEE Computer Society,Washington, DC, USA, 2009.
8. R.-K. Doong and P. G. Frankl. The ASTOOT Approach to Testing Object-Oriented Programs. ACM Trans. on Soft. Engineering and Methodology, 3, 1994.
9. Li N, Offutt J. Test Oracle Strategies for Model-based Testing[J]. IEEE Transactions on Software Engineering, 2017, PP(99):1-1.
10. Arantes A O, Santiago V A D, Vijaykumar N L. On Proposing a Test Oracle Generator Based on Static and Dynamic Source Code Analysis[C]// IEEE International Conference on Software Quality, Reliability and Security - Companion. IEEE, 2015:144-152.
11. Goffi A. Automatic generation of cost-effective test oracles[C]// ICSE '14: International Conference on Software Engineering. 2014:678-681.
12. Guo H F. A semantic approach for automated test oracle generation[J]. Computer Languages Systems & Structures, 2016, 45:204-219.
13. S. R. Shahamiri, W. M. N. W. Kadir, S. Ibrahim, S. Z. M. Hashim, An automated framework for software test oracle, Information and Software Technology 53 (7) (2011) 774–788.
14. S. R. Shahamiri, W. M. Wan-Kadir, S. Ibrahim, S. Z. M. Hashim, Artificial neural networks as multi-networks automated test oracle, Automated Software Engineering 19 (3) (2012) 303–334.
15. F. Wang, L.-W. Yao, J.-H. Wu, Intelligent test oracle construction for reactive systems without explicit specifications, in: Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on, IEEE, 2011, pp. 89–96.
16. A. Singhal, A. Bansal, et al., Generation of test oracles using neural network and decision tree model, in: Confluence The Next Generation Information Technology Summit, 2014 5th International Conference-, IEEE, 2014, pp. 313–318.
17. Farshad Gholami, Niousha Attar, Hassan Haghighi, Mojtaba Vahidi Asl, Meysam Valueian and Saina Mohamadyari. A Classifier-based Test Oracle for Embedded Software, 2018 Real-Time and Embedded Systems and Technologies (RTEST), pp. 104–111.
18. <http://sir.unl.edu/portal/index.php> (Accessed January 2017).
19. AndreasZeller.1999.Yesterday, My Program Worked. Today, It Does Not. Why?. InProc.joint meeting of the European Software Engineering Conference and theACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE’99). 253–267.

附中文参考文献:

1. [张震宇](http://xueshu.baidu.com/s?wd=authoruri%3A%28a24b31aef15ec17%29%20author%3A%28%E5%BC%A0%E9%9C%87%E5%AE%87%29%20%E9%A6%99%E6%B8%AF%E5%A4%A7%E5%AD%A6%E8%AE%A1%E7%AE%97%E6%9C%BA%E7%A7%91%E5%AD%A6%E7%B3%BB&tn=SE_baiduxueshu_c1gjeupa&ie=utf-8&sc_f_para=sc_hilight%3Dperson&sort=sc_cited) , [陈荣光](http://xueshu.baidu.com/s?wd=authoruri%3A%28da519ec48a3cffd9%29%20author%3A%28%E9%99%88%E8%8D%A3%E5%85%89%29%20%E9%A6%99%E6%B8%AF%E5%9F%8E%E5%B8%82%E5%A4%A7%E5%AD%A6%E7%94%B5%E8%84%91%E7%A7%91%E5%AD%A6%E7%B3%BB&tn=SE_baiduxueshu_c1gjeupa&ie=utf-8&sc_f_para=sc_hilight%3Dperson&sort=sc_cited) , [谢俊谦](http://xueshu.baidu.com/s?wd=authoruri%3A%2832f22b67a8073219%29%20author%3A%28%E8%B0%A2%E4%BF%8A%E8%B0%A6%29%20%E6%8B%9B%E5%95%86%E9%93%B6%E8%A1%8C%E9%A6%99%E6%B8%AF%E5%88%86%E8%A1%8C&tn=SE_baiduxueshu_c1gjeupa&ie=utf-8&sc_f_para=sc_hilight%3Dperson&sort=sc_cited) , [胡佩锋](http://xueshu.baidu.com/s?wd=authoruri%3A%28e39bc11a6d83954%29%20author%3A%28%E8%83%A1%E4%BD%A9%E9%94%8B%29%20%E6%8B%9B%E5%95%86%E9%93%B6%E8%A1%8C%E9%A6%99%E6%B8%AF%E5%88%86%E8%A1%8C&tn=SE_baiduxueshu_c1gjeupa&ie=utf-8&sc_f_para=sc_hilight%3Dperson&sort=sc_cited) 等.蜕变测试和断言检查的比较与实验研究[J]. 软件学报, 2009, 20(10):2637-2654.

1. \* 基金项目: 国家自然科学基金(Program No. 61103003),航天基金（基于大数据的航天嵌入式软件缺陷发现技术研究）

   Foundation item: National Natural Science Foundation of China (Program No. 61103003), Aerospace Fund (Research on defect discovery technology for aerospace embedded software based on big data)

   收稿时间: 0000-00-00; 修改时间: 0000-00-00; 采用时间: 0000-00-00; jos在线出版时间: 0000-00-00

   CNKI在线出版时间: 0000-00-00 [↑](#footnote-ref-1)