文章编号：2095-2163(2020)07-0075-07 中图分类号：TP311.53 文献标志码：A

基于分支路径与变量状态差异的软件缺陷定位方法

张大全，赵逢禹，刘 亚
（上海理工大学 光电信息与计算机工程学院，上海 200093）

摘 要：传统的基于测试的分支路径差异分析的软件缺陷定位方法不能有效利用程序源代码中变量状态等可用信息。针对此问题，本文提出了一种基于分支路径与变量状态差异分析的缺陷定位方法。首先对需要输出的分支和变量信息进行插桩，再使用 GCC 编译器对插桩后的程序进行编译输出，得到分支覆盖矩阵和变量值信息。使用余弦相似度公式获取相似分支路径对，并对相似分支路径对中的变量进行差异分析，通过变量差异分析来提高缺陷语句的可疑度。通过在 Siemens 套件上进行实验分析并与 DStar 等经典实验进行对比，基于分支路径与变量状态差异分析的方法能够有效提高软件缺陷语句怀疑度。

关键词：分支路径差异；变量状态差异；分支覆盖矩阵；相似分支路径对；软件缺陷定位

Software defect localization based on branches and variable state differences

ZHANG Daquan, ZHAO Fengyu , LIU Ya
(School of Optical-Electrical and Computer Engineering, University of Shanghai for Science and technology, Shanghai 200093, China)

【Abstract】The traditional software defect localization method based on branch paths difference analysis can not effectively explore the available information such as variable states when the program executed. To solve the problem, this paper proposes a defect localization method based on the analysis of the differences between branch paths and variable states. First, an instrumentation method is given for branch and variable information, and then the GCC compiler is used to compile the instrumented program and output the branch and variable information to obtain the branch coverage matrix. Then the cosine similarity formula to obtain similar branch path pairs is employed, and the differences in the variables in the similar branch path pairs to improve the suspiciousness of the defect statements are analyzed. Through the experimental analysis on the Siemens suite and comparison with classic method such as DStar, it is verified that the method based on the analysis of the differences between branch paths and variable states can effectively improve the suspicion of software defect statements.

【Key words】Branch Path Difference; Variable State Difference; Statement Coverage Matrix; Similar Branch Path Pairs; Software Defect Localization

0 引 言

软件缺陷定位是识别程序中故障位置的行为，被广泛认为是程序调试中最繁琐、最耗时、最昂贵但也是最重要的活动之一[1]。随着软件规模的扩大和复杂度的上升，人工进行软件缺陷定位的难度不断加大[2]，如何有效定位软件中存在的缺陷成为软件工程领域研究的热点。

软件缺陷的出现受多种因素影响，如软件规模、软件复杂度和开发人员的经验等[3]。传统的缺陷定位方法通常采用人工进行断点调试的方式，该方法困难且耗时，并且过程中可能受开发人员经验的影响。而软件缺陷自动化定位技术可由计算机自动查找软件中存在缺陷的位置，提高定位效率，可以排除人工经验的影响，保证软件可靠性。

在软件缺陷自动化定位相关研究中，以程序是否执行为标准，可将已有的软件缺陷定位方法分为静态方法和动态方法。静态方法不需要运行程序，只关注程序代码的文本信息，从中提取出代码结构和调用关系等信息进行分析。Weiser 等人提出静态切片的方法，该方法通过去除与错误无关的代码片段以缩小代码审查范围来帮助开发人员进行软件缺陷定位[4]；Moreno 等人针对面向对象语言，提取代码中的类名、方法名和注释等信息，利用文本检索的方法统计它们在缺陷报告中出现的频率来进行定位[5]。由于静态的软件故障定位方法仅静态分析程序的数据流和控制流程，这类方法所利用的程序代码信息有限，定位代码缺陷的精度较低[6]。

动态方法需要运行程序测试用例，收集程序执

基金项目：国家自然科学基金(61803264)。

作者简介：张大全(1993-)，男，硕士研究生，主要研究方向：软件缺陷定位；赵逢禹(1963-)，男，博士，教授，硕士生导师，主要研究方向：计算机软件与软件系统安全、软件工程与软件质量控制；刘 亚(1983-)，女，博士，副教授，硕士生导师，主要研究方向：信息安全、密码学、区块链等。

收稿日期：2020-04-21

行轨迹和覆盖语句等信息进行定位。基于程序测试用例执行覆盖频谱,Jones 等人提出 Tarantula 怀疑度公式,他们认为执行失败的测试用例覆盖的语句怀疑度要高于执行成功的测试用例覆盖语句,并给出计算公式[7]。随后又有人提出了不同的怀疑度计算公式,如 Jaccard、Ochiai 和 DStar 公式,其中由 Wong 等人提出的 DStar 方法被证明要优于其它三种公式算法;Renieres 等人基于程序执行路径差异,提出了最近邻模型,该模型以测试用例执行路径样本的距离作为不相似度的衡量标准,分析当前执行失败的路径与邻居路径之间的差异来定位缺陷[8];刘丹凤等人以函数为基本粒度,对程序执行时函数的调用路径进行分析,挖掘缺陷之间的关联,来缩小缺陷定位范围并定位缺陷所在[9];杨波等人提出了基于数据链的故障定位方法,利用变量之间的调用关系进行缺陷定位[10]。

尽管采用基于测试覆盖频谱的动态方法在软件缺陷定位中取得了不错的效果,但这些方法仍然存在以下问题:

(1)可疑语句排名报告中相邻的语句在程序中大多没有关联性,按照排名语句检查缺陷时需要检查比当前排名语句位置更多的代码;

(2)没有充分利用程序执行过程中的信息,如测试用例执行过程中变量状态的相关信息。

基于上述研究,本文认为在测试用例执行过程中,除获得语句执行频谱信息外,还可以提取测试用例执行路径上的变量状态变化信息。实验分析证明,本文提出的基于分支与变量状态差异的缺陷定位方法可以使缺陷报告中相邻语句之间更具关联性,同时将存在缺陷的语句排在可疑度排名靠前的位置,从而提高定位的精确度和效率。

1 研究动机

表 1 为一个经典的启发性程序示例,该程序的功能为求三个数的最大值。将程序的代码行号标记为 S_1 至 S_{19},其中 S_{12} 为故障所在位置,正确语句应为:$if(a[i] > max)$。针对这个比较大小的程序,本文设计了 5 个测试用例,分别为 t_1、t_2、t_3、t_4 和 t_5。

测试用例执行的结果 result 用数字 0 和 1 来表示,0 表示测试用例执行未通过,1 表示通过。测试用例在执行过程中是否执行了某条语句亦用 0 和 1 来表示,0 表示未执行,1 表示测试用例执行时经过该语句。从表 1 中可以看出,在 5 个测试用例的执行过程中,程序中的每一条可执行语句都被执行覆盖了。

表 1 中的可疑度是 Tarantula 方法对该程序示例进行定位的结果。根据 Tarantula 算法公式可得,所有语句存在缺陷的可疑度都是相同的,找到的缺陷可疑语句集合为整个程序的可执行语句,因此无法有效进行缺陷定位。同理,Jaccard、Ochiai 和 DStar 等利用统计学方法进行概率计算的公式在这种情况下都会失效。尽管该特例在计算语句可疑度时不具有统计意义,但实际上基于路径的语句执行频谱分析的缺陷定位确实存在执行分支相同,分支间不存在覆盖差异的情况。根据传统的分支路径分析定位不准确的问题,本文认为,可以从程序中挖掘更多的可用信息以进行缺陷定位。如在示例程序中,所有测试用例执行覆盖的程序语句都是相同的,但 t_2、t_5 执行结果(表 1 中 *result*)通过,t_1、t_3、t_4 执行未通过。由覆盖语句相同但执行结果不同这一信息可知,不同的测试用例在执行过程中,必有除语句覆盖信息之外的其它信息差异,导致其运行结果不同。

从人工进行软件缺陷定位的角度出发,开发人员在进行断点调试的过程中,除观察语句执行的覆盖情况,通常还会观察程序中定义的变量状态是否改变。表 1 展示的示例中共有两个变量,i 和 max。在输入不同测试用例的情况下,变量 i 的取值状态变化都相同,但 max 的取值状态变化有明显的差异,而 max 与错误语句所在位置 S_{12} 有直接的关联性。因此本文认为,将程序中的变量状态信息提取出来加以分析,能够有效地提高软件缺陷定位的准确度。

2 实现方法

本文采用分支路径与变量状态差异结合的缺陷定位分析方法,该方法以测试用例执行分支作为程序执行路径的基本单位,使用余弦公式分别对测试用例执行成功的和执行失败的路径进行相似度计算,并提取相似的路径对。对相似路径对中涉及到的变量进行差异分析,最终定位程序中缺陷的位置。

具体实现方法有以下四步:

(1) 确定待提取分支和变量信息;

(2) 代码插桩并执行测试用例;

(3) 分支覆盖矩阵生成;

(4) 相似路径对选择和缺陷语句定位。

2.1 确定待提取分支和变量信息

使用源码级插桩的方法收集测试用例执行经过的分支和变量的动态变化信息,首先要确定待输出的分支和变量信息。通过静态文本分析的方法,对程序源代码进行文本分析,提取出代码中的分支和

变量信息。对于分支信息，需要获取分支的位置以及测试用例执行时是否经过该分支，因此需提取分支的位置和分支中的谓词表达式。对于变量，则需要提取程序中变量赋值语句位置信息和被赋值的变量名信息。

以 C 语言为例，对于分支信息，使用文本匹配的方法，将代码中含有 if、else、switch 关键词的语句进行提取，并使用三元组 (L,S,P) 表示和存储，L 表示分支行号，S 表示分支语句，P 表示分支中的谓词表达式。同理可提取出程序源码中的变量信息，对于变量，使用二元组 (L,N) 表示，L 表示变量赋值语句所在行号，N 表示变量名。

表 1　基于 Tarantula 方法的故障定位举例

Tab. 1　An example of fault localization based on Tarantula method

程序	行号	$t_1(5,4,3)$	$t_2(5,5,5)$	$t_3(5,3,4)$	$t_4(3,5,4)$	$t_5(3,3,3)$	可疑度
#include<stdio.h>	s_1	–	–	–	–	–	–
#include<stdlib.h>	s_2	–	–	–	–	–	–
#define N 3	s_3	–	–	–	–	–	–
int main( )	s_4	1	1	1	1	1	0.6
{	s_5	–	–	–	–	–	–
int i;	s_6	1	1	1	1	1	0.6
int a[N];	s_7	1	1	1	1	1	0.6
int max;	s_8	1	1	1	1	1	0.6
max = a[0];	s_9	1	1	1	1	1	0.6
for(i=0;i<N;i++)	s_{10}	1	1	1	1	1	0.6
{	s_{11}	–	–	–	–	–	–
if(a[i]<=max)	s_{12}	1	1	1	1	1	0.6
{	s_{13}	–	–	–	–	–	–
max=a[i]	s_{14}	1	1	1	1	1	0.6
}	s_{15}	–	–	–	–	–	–
}	s_{16}	–	–	–	–	–	–
printf("max: %d\n",max);	s_{17}	1	1	1	1	1	0.6
return 0;	s_{18}	1	1	1	1	1	0.6
}	s_{19}	–	–	–	–	–	–
result	–	0	1	0	0	1	–
变量状态	–	–	–	–	–	–	–
i	–	0,1,2	0,1,2	0,1,2	0,1,2	0,1,2	–
max	–	5,4,3	5,5,5	5,3,3	3,3,3	3,3,3	–

2.2　代码插桩并执行测试用例

已获取到程序源码中待提取分支和变量的相关信息，使用源码级插桩的方法将这些信息的输出语句在程序源码中插入，以便在测试用例执行时将它们动态输出。对于分支信息，可在程序源码中出现分支代码的语句前插入输出语句，将该分支中谓词表达式的结果输出，如果执行测试用例时该谓词表达式输出为真，表示分支将被执行，反之，表示分支并未执行；对于变量状态信息，则需要在程序源码中变量赋值语句后插入输出语句，输出变量赋值操作后的变量值，来收集整个程序中各个变量值的变化信息。

以表 1 中的程序及其测试用例 t_1 为例。执行测试用例后，收集到的分支和变量信息见表 2 和表 3。

表 2　分支执行信息

Tab. 2　Branch execution information

行号	分支	分支是否执行
10	for(i=0;i<N;i++)	是
12	if(a[i]<=max)	是

表 3 变量取值信息

Tab. 3 Variable value information

行号	变量	取值
10	i	0
14	max	5
10	i	1
14	max	4
10	i	2
14	max	3

2.3 分支覆盖矩阵生成

为便于对不同测试用例执行经过的分支路径进行相似度计算,将程序每一个测试用例执行经过的分支和结果用向量 PV_i 表示为式(1)。

$$PV_i = [P_1, \quad P_2, \quad \dots \quad P_m, \quad R_i], \tag{1}$$

其中, PV_i 表示分支覆盖与结果向量; P_m 表示测试用例执行时是否经过该分支,经过用 1 表示,未经过用 0 表示; R_i 表示测试用例的执行结果,执行通过用 1 表示,未通过用 0 表示。

全部测试用例执行后,经过的分支和测试用例执行的结果用矩阵 BM 表示,式(2)。

$$BM = [PV_1, PV_2, \dots PV_n]^T. \tag{2}$$

其中, BM 表示分支覆盖矩阵,为一个 $m \times n$ 的矩阵,矩阵的行数 m 由程序中的分支数决定,矩阵的列数 n 由测试用例数量决定。

在表 1 的程序中,其分支覆盖矩阵 BM 如下所示:

$$BM = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix}^T$$

矩阵 BM 中的向量 $PV_1 = [1,1,0]^T$ 表示测试用例 t_1 执行时经过两个分支,其执行结果未通过。

2.4 路径相似度计算和缺陷语句定位

传统的基于路径差异的缺陷定位方法通过算法计算,分别在执行成功的测试用例经过的路径集合中和执行未通过的路径集合中各选出一条代表路径,进行差异对比,但这种方法过于依赖路径的选取。针对此问题,本文对执行失败的路径集合中的每条路径分别在执行成功的路径集合中选取多条相似的路径,组成相似分支路径对集合,对每一个路径对进行变量差异分析。本文选取余弦相似度大于 0.8的路径对作为相似分支路径对进行分析。

夹角余弦公式(3)如下:

$$similarity(x_i, y_i) = \frac{\sum_{i=1}^{n} (x_i \times y_i)}{\sqrt{\sum_{i=1}^{n} (x_i)^2} \times \sqrt{\sum_{i=1}^{n} (y_i)^2}}. \tag{3}$$

相似分支路径对集合以公式(4)表示:

$$sbp = \{(PV_f, PV_s) \mid similarity(PV_f, PV_s) \geq 0.8\}, \quad PV_f \in BM_f, PV_s \in BM_s. \tag{4}$$

其中, sbp 表示相似分支路径对集合,矩阵 BM_f 表示执行失败的测试用例路径集合,矩阵 BM_s 表示执行成功的测试用例路径集合,向量 PV_f 表示执行失败的测试用例路径,向量 PV_s 表示执行成功的测试用例路径。

对相似分支路径对中两条路径上的变量进行差异对比,以执行未通过的分支路径为基准,将执行通过的路径上与之存在差异的变量涉及到的语句怀疑度提升。

以表 1 中的程序为例,表 4 为分支路径相似度值,其中 t_2、t_5 为通过的测试用例,t_1、t_3、t_4 为未通过的测试用例。

表 4 分支路径相似度

Tab. 4 Similarity of branches

测试用例	t_2	t_5
t_1	1	1
t_3	1	1
t_4	1	1

需要说明的是,由于表 1 中的程序具有一定特殊性,即所有测试用例经过的分支路径都相同,计算得到的分支路径相似度都为 1,但这并不影响本文方法的使用。

首先,应用公式(4)得到相似分支路径对: $\{t_1, t_2\}$, $\{t_1, t_5\}$, $\{t_3, t_2\}$, $\{t_3, t_5\}$, $\{t_4, t_2\}$, $\{t_4, t_5\}$;其次,对每个路径对中的变量取值做差异比较。由表 1 中倒数第 3 行至倒数第 1 行的变量状态数据可知,对于示例中的相似分支路径对,除 $\{t_4, t_5\}$ 外,路径对中的变量 max 取值状态均存在差异。因此,假设每条语句的初始怀疑度为 0,则将变量 max 所在语句的可疑度提高 5,此时,语句 S_9、S_{12}、S_{14} 的可疑度为 5,其余语句可疑度为 0,而 S_{12} 语句为缺陷所在语句,从而证明该方法可有效定位到缺陷语句。

3 算法实现

在分支路径与变量状态差异结合的缺陷定位分析方法中,主要有两个算法:

(1)分支与变量值的获取和分支覆盖矩阵生成算法,运行插桩代码获取分支执行信息后,该算法进一步通过测试用例执行经过的分支信息和测试用例执行结果来生成分支覆盖矩阵;

(2)可疑语句排名算法,通过对分支覆盖矩阵进行分支路径相似度计算和变量值差异分析,调整

语句可疑度分数，最终给出缺陷语句可疑度排名。

3.1　分支覆盖矩阵生成算法

本文以 C 语言源码作为研究对象，研究结论对其它语言也有适用性。使用 GCC 编译器对插桩后的程序源码进行编译执行，获得了程序分支的执行覆盖信息和执行结果，算法 1 通过三个处理步骤，可将这些文本信息转换为矩阵的形式，供算法 2 进行进一步处理。

算法 1：分支覆盖矩阵生成算法

输入：分支执行文本信息

输出：分支覆盖矩阵

(1)根据分支和测试用例数量创建 m×n 的空矩阵 BranchMatrix；

(2)读取分支执行文本文件，根据分支的执行情况填充矩阵元素，执行为 1，未执行为 0；

(3)读取测试用例执行结果文件，将执行通过的测试用例对应矩阵中的元素值置为 1，未通过的置为 0。

3.2　可疑语句排名算法

算法 1 将代码插桩并执行测试用例后输出的分支覆盖信息转换为矩阵的形式，在算法 2 中，根据测试用例执行的结果，进一步把分支覆盖矩阵划分为执行通过的和执行未通过的两个分支覆盖矩阵，以测试用例执行未通过的矩阵中的路径为基础，将其中的每条路径与执行通过的矩阵中的每条路径做夹角余弦值运算，取结果值大于 0.8 的路径作为相似分支路径对。利用插桩输出的变量值信息，在每个代表路径对中做变量值差异比较，根据变量差异信息，提高差异变量所在的语句的可疑度。

算法 2 ：分支相似度计算与可疑语句排名算法

输入：分支覆盖矩阵、变量执行信息

输出：语句可疑度排名

(1)根据测试用例执行是否通过，将矩阵划分为测试用例执行通过的矩阵 SuccMatrix 和执行未通过的矩阵 FaultMatrix；

(2)依次取 FaultMatrix 中的路径，分别与 SuccMatrix 中的路径做夹角余弦值计算，得到分支相似度 BranchSimilarity；

(3)取 BranchSimilarity 大于 0.8 的分支，组成相似分支路径对；

(4)对每一个相似分支路径对比较路径中的变量是否存在差异；

(5)对于存在差异的变量 DiffVariable，将涉及该变量的语句 DiffStatement 的可疑度增加 1。

(6)对每个相似分支路径对重复执行步骤(4)和(5)，累计 DiffStatement 的可疑度，并将可疑度由高到底排序，最终得到语句可疑度排名 StatementRank。

4　实验分析

本文设计了对比实验，将本文提出的方法与 Tarantula、Ochiai 和 DStar 三种经典方法在同一数据集上进行定位实验，通过对比相同代码检测率下定位到的缺陷比率来说明本文方法的有效性。

4.1　实验数据

本文使用经典的 Siemens 数据集进行实验，该实验套件包含 tcas、replace 等 7 个 C 语言编写的程序，每个程序包中都包含一个正确程序版本和若干被人工植入错误的版本，利用其中的错误版本和自带的测试用例进行实验，来验证本文方法的有效性。该数据集的所有版本源码及测试用例都可以在 SIR 网站获得。表 5 为本实验程序的基本信息。

表 5　Siemens 程序基本信息

Tab. 5　Siemens programs basic information

程序名	版本	代码	测试用例数	用例描述
printtokens	7	472	4056	词法分析
printtokens2	10	399	4071	词法分析
replace	32	512	5542	模式匹配替换
schedule	9	292	2650	优先级调度
schedule2	10	301	2680	优先级调度
tcas	41	141	1578	高度划分
totinfo	23	440	1054	统计多表数据

4.2　实验配置及步骤

本文实验平台为 CentOS7 系统，对应的 GCC 版本为 4.8.5。实验具体流程如下：

(1)使用本文编写的插桩工具对程序源码进行插桩；

(2)使用 GCC 编译器编译执行插桩后的程序源码。得到测试用例执行的分支与变量信息和测试用例执行结果；

(3)使用根据算法 1 编写的工具程序将分支执行信息转化为分支覆盖矩阵；

(4)划分分支覆盖矩阵并使用夹角余弦公式计算分支相似度，筛选相似路径对；

(5)对相似路径对中的变量做差异分析，并计算语句可疑度；

(6)将本文方法与 Tarantula、Ochiai 和 DStar 方法得到的语句可疑度排名进行对比。

4.3　实验结果及分析

实验首先以 Siemens 数据集中的 tcas 程序为例

说明本文方法的定位过程和效果,展示在整个数据集上进行实验,并对其进行分析。对 tcas 程序的 v12 错误版本执行上述实验步骤,得到如下实验中的中间数据。

以测试用例 t_3 为例,运行插桩后的程序源码并执行测试用例,可得到表6所示分支执行信息和表7所示变量取值信息。

表 6 测试用例 t3 分支执行信息

Tab. 6 Branch execution information

行号	分支语句	是否执行
124	if(enabled &&...)	是
73	if(upward_preferred)	是
91	if(upward_preferred)	是

表 7 变量取值信息

Tab. 7 Variable value information

行号	变量名称	取值
118	enabled	1
119	tcas_equipped	0
120	intent_not_known	1
122	alt_sep	0
72	upward_preferred	1
...	...	...

通过对生成的分支覆盖矩阵执行余弦相似度计算,得到表8所示分支相似度,其中 t_1、t_2、t_5、t_{10} 等为执行通过的测试用例,t_7、t_{80} 等为执行未通过的测试用例。

表 8 tcas 程序版本 v12 分支相似度

Tab. 8 Branch similarity of TCAS v12

测试用例	t_7	t_{80}	...
t_1	1	1/5	...
t_2	3/4	1/4	...
t_5	1/4	1/2	...
t_{10}	1/5	1	...
...	...	...	...

tcas 程序的 v12 版本共执行了1 578个有效测试用例,已知该版本的缺陷语句存在于 S_{118}。表 9 为经过计算最终得到的语句可疑度排名。

表 9 缺陷语句可疑度排名

Tab. 9 Ranking of suspect degree of defective statements

算法	可疑缺陷语句排序
本文算法	S_{124} S_{118} S_{114} S_{120} S_{119}...
Tarantula	S_{154} S_{153} S_{152} S_{151}... S_{118}...
Ochiai	S_{154} S_{153} S_{152} S_{151}... S_{118}...
DStar	S_{154} S_{153} S_{152} S_{151}... S_{118}...

由表 9 可以看出,本文试验方法将故障语句排在第二位,而其它三种方法均将缺陷语句排在前五之外。结合 tcas 程序源码,观察本文方法中得到的语句排名可以发现,缺陷所在语句 S_{118} 与其前后的语句 S_{124} 和 S_{114} 都与程序中定义的变量“enabled”相关,语句位置分布相对集中,表明本文方法计算得出的可疑缺陷语句间有较强的关联性,便于实际应用中开发人员进行缺陷定位。

本文对 Siemens 套件中的其它程序也进行了相同实验,定位的结果如图 1 所示。在折线图中,X 轴表示检测的代码语句的百分比;Y 轴表示定位缺陷的百分比;图中的节点表示代码检测率在某百分比下,能够定位出缺陷的百分比。

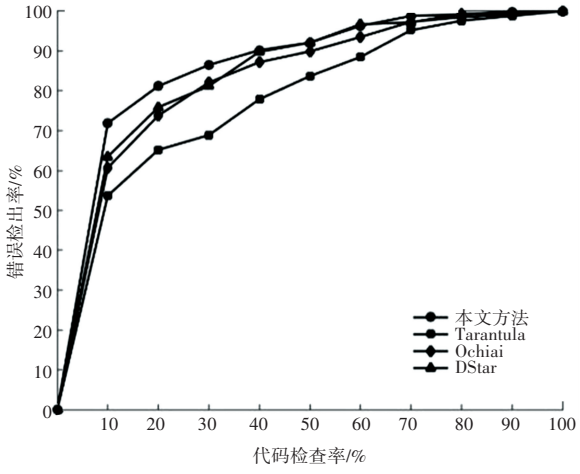

图 1 Siemens 程序定位结果对比

Fig. 1 Comparison of Siemens program localization results

从本文方法与其它三种对比方法的定位效率折线图中可以看出,在 Siemens 程序集中,应用本文方法检查 10%的代码可以有效定位 71.9%的缺陷,而其它三种方法分别是:Tarantula 方法定位到 53.7%的缺陷,Ochiai 方法定位到 60.7%的缺陷,Dstar 方法定位到 63.5%的缺陷。这就意味着在实际的缺陷定位中,在同样定位到缺陷的情况下,使用本文缺陷定位方法开发人员需要检查的代码量更少,进行缺陷定位的效率更高。

5 结束语

本文提出了一种基于分支与变量状态差异的软件缺陷定位方法,首先使用 Python 语言编写插桩工具,结合 GCC 编译器获得程序运行时的分支与变量状态变化信息,在此基础上提出新的可疑语句排名方法,实现了程序缺陷语句的定位,提高了定位精确度。使用 Siemens 程序集作为实验数据集,通过对比实验证明了该方法的有效性。

在实验中，缺陷版本中的错误为人工植入，不能够完全代表实际开发中存在缺陷的情况，在后续工作中还需在更大的样本集上进行大量的实证研究，以验证本文方法在实际软件开发中缺陷定位的效果。

参考文献

[1] Wong，W. Eric，GAO Ruizhi，LI Yihao，et al. A Survey on Software Fault Localization[J]. IEEE Transactions on Software Engineering，2016：707-740.

[2] VESSEY I . Expertise in Debugging Computer Programs[J]. Social Science Electronic Publishing，2008，23(5)：459-494.

[3] 哈清华，姜瑞凯，刘逻.软件缺陷的生成因素分析[J]. 计算机技术与发展，2016，1(1)：1-5.

[4] WEISER M . Program Slicing[J]. IEEE Transactions on Software Engineering，1984，10(4)：352-357.

[5] MORENO，LAURA，BANDARA，et al. On the Relationship between the Vocabulary of Bug Reports and Source：Code[C]// IEEE International Conference on Software Maintenance. IEEE Computer Society，2013.

[6] WEN W . Software fault localization based on program slicing spectrum[C]//International Conference on Software Engineering. IEEE Press，2012.

[7] JONES J A. Empirical evaluation of the tarantula automatic fault-localization technique[C]//IEEE/ACM International Conference on Automated Software Engineering. 2005.

[8] RENIERES M ，REISS S P . Fault localization with nearest neighbor queries[C]//18th IEEE International Conference on Automated Software Engineering，2003. Proceedings. IEEE，2003.

[9] 刘丹凤，牟永敏. 基于函数调用路径关联分析的缺陷定位方法研究[J]. 计算机应用研究 2016(8)：2363-2370.

[10] 杨波，吴际，刘超. 基于数据链的软件故障定位方法[J]. 软件学报，2015(2)：254-268.

(上接第 74 页)

$$LF1 = \begin{bmatrix} -0.085\,8 & -0.019\,5 \\ -0.001\,8 & -0.033\,6 \end{bmatrix};$$

$$LF2 = \begin{bmatrix} 0.049\,9 & 0.017\,2 \\ 0.011\,86 & 0.011\,8 \end{bmatrix};$$

变时滞 $\tau_i = 0.3 + 0.1\cos t$；

设马尔可夫转移概率为：

$$\Pi = \begin{bmatrix} -4 & 4 \\ 3 & -3 \end{bmatrix}$$

通过仿真可得 Markov 链的状态，如图 1 所示；设初值：

$$x^{\mathrm{T}}(0) = [2 \quad -1];$$

$$\hat{x}^{\mathrm{T}}(0) = [2.5 \quad -1.5];$$

以及外部干扰信号：

$$v(t) = 0.2exp(0.3t). \tag{15}$$

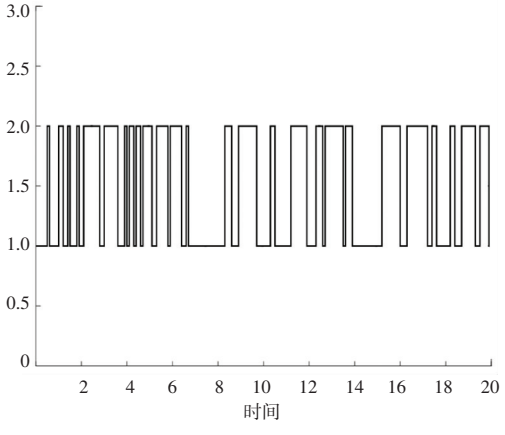

图 1　跳变模态

Fig. 1　Transition mode

图 2 表示系统(6)的误差曲线；在规定时间内快速收敛为 0，保证系统的稳定性。以上仿真结果表明所设计的滤波器能够达到预期。

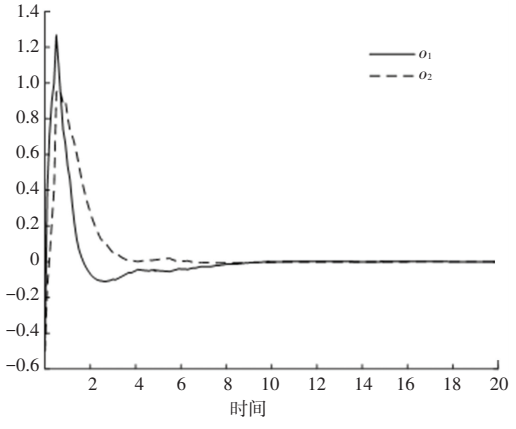

图 2　滤波器系统误差图

Fig. 2　Error diagram of the filter system

4　结束语

本文研究了一种带马尔科夫跳变的中立型混合时滞系统的 H_∞ 滤波。首先，构造了一种新型李亚普诺夫方程，利用积分不等式方法在证明过程中对积分行进行的特殊处理来消除积分项，从而降低了系统的决策量与计算的复杂度。最后，通过数值举例的方法说明本文方法的有效性。

参考文献

[1] Elsayed and M. J. Grimble A new Approach to H∞ design of optimal digital filters[J]. IMAJ Math. Control Information，1989，6(81)：233-251.

[2] 陈绍东，杜书德，仝云旭. 奇异中立系统降维 H∞ 滤波器的设计[J]. 吉林大学学报(理学版)，2017，55(5)：1151-1157.