(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 110515826 A
(43)申请公布日 2019.11.29

(21)申请号 201910594565.X

(22)申请日 2019.07.03

(71)申请人 杭州电子科技大学
地址 310018 浙江省杭州市下沙高教园区2号大街

(72)发明人 王兴起 程瑞洁 陈滨 魏丹 方景龙

(74)专利代理机构 杭州君度专利代理事务所（特殊普通合伙）33240
代理人 杨舟涛

(51)Int.Cl.
G06F 11/36(2006.01)
G06N 3/08(2006.01)

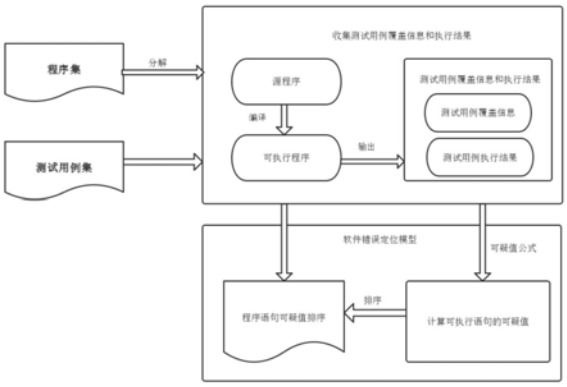
权利要求书1页 说明书4页 附图5页

(54)发明名称

一种基于次数频谱与神经网络算法的软件缺陷定位方法

(57)摘要

本发明公开了一种基于次数频谱与神经网络算法的软件缺陷定位方法，本发明所提出的方法考虑了由于循环等过程的存在引起语句多次执行，使用程序分支计数频谱对覆盖信息进行扩展，并将其作为BP神经网络的输入，实现运用机器学习的方法实现软件缺陷定位的目标。基于覆盖的程序缺陷定位方法利用覆盖信息来定位程序的缺陷语句。然而，这些方法可能会受到代码偶然性正确性的不利影响。偶然性正确是指在执行缺陷语句后，并没有触发错误的测试用例执行过程。本方法将会发生偶然性正确的测试用例的执行结果统一修正为错误的方法，在最大程度上避免偶然性正确对程序定位性能的影响。



CN 110515826 A

1.一种基于次数频谱与神经网络算法的软件缺陷定位方法,其特征在于,具体包括以下步骤:

步骤一:执行正确的测试用例,并得到执行结果;

在不同待测程序中分别包含有正确的测试用例,运行完正确的测试用例并保存相应的执行结果;

步骤二:执行所有的包含不同错误的测试用例集,并保存结果;

不同的待测程序中分别包含有各自带有不同错误的测试用例集,执行所有的错误测试用例集,得到对应的结果;

步骤三:根据程序运行,收集程序分支触发频谱和程序分支计数频谱;

在Ubuntun系统中使用GCC对源程序进行编译处理,然后使用Gcov工具收集测试用例的语句覆盖信息和执行结果,得到程序分支触发频谱和程序语句计数频谱;其中程序分支触发频谱收集程序语句是否执行,程序分支触发频谱收集程序语句执行的次数;

步骤四:把收集到的程序频谱输入神经网络中,训练得到一个稳定的神经网络模型;

步骤五:设置虚拟单位矩阵,输出每条语句的可疑值;

在神经网络中设置一个等同于程序语句的单位矩阵作为虚拟矩阵,得到每条语句的可疑值;

步骤六:排除偶然性正确测试用例的干扰;

在成功测试用例集中查找与失败测试用例语句覆盖信息完全一致的测试用例,称之为偶然性正确测试用例,把偶然性正确测试用例的执行结果改为失败,或者删除偶然性正确测试用例;

步骤七:根据EXAM值对程序语句进行排序,从高到低依次检查程序语句是否包含错误,直到找到错误语句。

一种基于次数频谱与神经网络算法的软件缺陷定位方法

技术领域

[0001] 本发明属于软件工程的自动化测试方法领域，涉及排除偶然性正确测试用例影响，基于次数频谱，运用神经网络算法，实现软件缺陷定位的方法，具体涉及一种基于次数频谱与神经网络算法的软件缺陷定位方法。

背景技术

[0002] 软件缺陷是软件产品中会导致软件失效的瑕疵。可以基于对软件本身代码进行统计分析推断软件缺陷，也可以基于软件频谱信息软件自动化测试技术主要研究点包括软件缺陷预测和软件缺陷定位两个方向。软件缺陷定位的目是准确确定缺陷的位置。软件缺陷定位是一项非常繁琐且耗时，被认为是程序调试中最昂贵的活动之一。软件缺陷的行为算法旨在帮助开发人员更高效地发现缺陷。例如，基于程序切片的缺陷定位方法使用程序片进行有效的统计缺陷定位，基于不变量的缺陷定位方法，基于谓词的缺陷定位方法。

[0003] 基于程序切片的缺陷定位方法可分为两个主要阶段：静态缺陷定位，动态缺陷定位。这些方法通过缩小搜索空间来定位缺陷。基于不变量的方法尝试在失败的程序执行中发现违反程序属性以定位缺陷。基于谓词的缺陷方法主要使用抽样方法在执行过程中收集谓词覆盖率数据。谓词根据其可疑分数划分优先级，这表明谓词和程序缺陷之间的关系。应首先检查具有较高分数的谓词，以帮助程序员找到缺陷。基于程序频谱的软件缺陷定位方法通过在某些方面记录程序的执行信息来跟踪程序行为，如代码的执行时间，分支或循环的信息。它可用于计算代码行的可疑值。基于程序频谱的软件缺陷定位方法是一种更有效的软件缺陷定位方法，因为没有必要考虑程序本身的内部结构而且执行成本低。基于程序频谱的软件缺陷定位方法是目前研究最为广泛的缺陷定位方法。程序频谱通常是指程序在执行期间代码覆盖信息的集合，它是程序动态行为特征的描述。程序频谱的收集通过以下3个步骤：首先，准备用于测试的测试用例集以及可执行的待测程序；然后，将测试用例集中的测试用例在待测程序中执行，并标记测试用例对程序语句的覆盖情况。最后，将测试用例的执行结果与预期结果进行比较，获得执行结果是否正确的标签信息。重复上述步骤，直到运行完全部测试用例，即可收集到程序语句执行覆盖信息以及测试用例的执行结果标签信息，获得覆盖信息表。覆盖信息表和一组测试用例标签可用于识别程序源代码的定量度量信息。程序语句根据可疑值进行优先排序，可通过覆盖率信息和标签计算。可疑值较高的代码行更有可能包含缺陷。基于频谱的故障定位方法的基本思想是一条语句的可疑值会随着被失败测试用例执行次数的增加而增加，与此相反，一条语句的可疑值会随着被成功测试用例执行次数的增加而减少。

发明内容

[0004] 本发明的目的在于克服现有方法的不足，提供一种基于次数频谱与神经网络算法的软件缺陷定位方法。本发明通过收集程序分支计数频谱信息，排除偶然性正确测试用例的干扰，并使用神经网络模型提高软件缺陷定位效率。

[0005] 本发明一种基于次数频谱与神经网络算法的软件缺陷定位方法,具体包括以下步骤:

[0006] 步骤一:执行正确的测试用例,并得到执行结果;

[0007] 在不同待测程序中分别包含有正确的测试用例,运行完正确的测试用例并保存相应的执行结果;

[0008] 步骤二:执行所有的包含不同错误的测试用例集,并保存结果;

[0009] 不同的待测程序中分别包含有各自带有不同错误的测试用例集,执行所有的错误测试用例集,得到对应的结果;

[0010] 步骤三:根据程序运行,收集程序分支触发频谱和程序分支计数频谱;

[0011] 在Ubuntun系统中使用GCC对源程序进行编译处理,然后使用Gcov工具收集测试用例的语句覆盖信息和执行结果,得到程序分支触发频谱和程序语句计数频谱;其中程序分支触发频谱收集程序语句是否执行,程序分支触发频谱收集程序语句执行的次数;

[0012] 步骤四:把收集到的程序频谱输入神经网络中,训练得到一个稳定的神经网络模型;

[0013] 步骤五:设置虚拟单位矩阵,输出每条语句的可疑值;

[0014] 在神经网络中设置一个等同于程序语句的单位矩阵作为虚拟矩阵,得到每条语句的可疑值;

[0015] 步骤六:排除偶然性正确测试用例的干扰;

[0016] 在成功测试用例集中查找与失败测试用例语句覆盖信息完全一致的测试用例,称之为偶然性正确测试用例,把偶然性正确测试用例的执行结果改为失败,或者删除偶然性正确测试用例;

[0017] 步骤七:根据EXAM值对程序语句进行排序,从高到低依次检查程序语句是否包含错误,直到找到错误语句。

[0018] 本发明较传统方法来说有以下显著优点

[0019] (1)突出了程序中由于循环等的存在程序语的执行次数对执行结果的影响。

[0020] (2)排除了偶然性正确测试用例对程序语句可疑值计算的干扰。

[0021] (3)使用神经网络代替传统的手工计算的方式。

[0022] 本发明采用的具体实现方法是:

[0023] (1)执行正确的测试用例,并得到执行结果;

[0024] (2)执行所有的包含不同错误的测试用例集,并保存结果;

[0025] (3)根据程序运行,收集程序分支触发频谱和程序分支计数频谱;

[0026] (4)排除偶然性正确测试用例的干扰;

[0027] (5)把收集到的程序频谱输入神经网络中,训练得到一个稳定的神经网络模型;

[0028] (6)设置虚拟单位矩阵,输出每条语句的可疑值;

[0029] (7)依据EXAM值对程序语句进行排序,从高到低依次检查程序语句是否包含错误,直到找到错误语句。

[0030] 有益效果:由本发明提出的方法,突出强调程序中由于循环等存在程序语句执行次数对测试用例执行结果的影响,并且排除偶然性正确测试用例对语句可疑值计算的干扰,使用神经网络模型代替传统的手工计算方式,使得程序员能够检查较少的程序语句就

能够定位到错误语句,提高了软件缺陷定位的效率。

附图说明

[0031] 图1是本发明的软件缺陷定位的流程图;

[0032] 图2是本发明的神经网络结构图;

[0033] 图3是本发明的实验结果综合示例图;

[0034] 图4是本发明神经网络的BHS和BCS与传统方法对比的示例图;

[0035] 图5是本发明修改偶然性正确测试用例执行结果与神经网络方法对比示例图;

[0036] 图6是本发明删除偶然性正确测试用例结果示例图。

具体实施方式

[0037] 下面结合实例附图进行进一步详细说明本发明的具体实现方法:

[0038] (1)执行正确的测试用例,并得到执行结果;

[0039] 实例是在Siemens Suite数据集上进行,Siemens Suite数据集包含了print_tokens、print_tokens2、Schedule、Schedule2、replace、tcas、tot_info 7个子数据集,分别执行每个数据集正确版本的源代码,并得到执行结果。

[0040] (2)执行所有的包含不同错误的测试用例集,并保存结果;

[0041] 在每个数据集中分别包含有不同错误版本的可执行代码,例如print_tokens包含有7个不同错误版本的源程序,每个错误版本中有4000多个测试用例。自动化执行完所有的测试用里,得到执行结果。

[0042] (3)根据程序运行,收集程序分支触发频谱和程序分支计数频谱;

[0043] 在Linux系统下,借助GCC工具,对源程序进行编译处理,然后,利用Gcov工具跟踪每条程序语句是否执行,以及每条语句执行了多少次。依次输出每个测试用例中语句的执行轨迹,以及最后执行结果标签,分别得到程序分支触发频谱和程序分支计数频谱。

[0044] 图1是整个的软件缺陷定位的流程图;源程序和测试用例作为收集程序分支触发频谱和程序分支次数频谱的基础。

[0045] (4)把收集到的程序频谱输入神经网络中,训练得到一个稳定的神经网络模型;

[0046] 把收集到的测试用例的程序分支触发频谱和程序分支计数频谱信息依次作为神经网络的输入向量。通过不断训练得到一个稳定的神经网络模型。

[0047] (5)设置虚拟单位矩阵,输出每条语句的可疑值;

[0048] 根据每个数据集的程序语句数目分别设置一个虚拟测试用例,依次输出每条语句的可疑值。

[0049] 图2为神经网络的结构图,输入神经元个数等于输入神经网络的程序频谱的语句数量,不断调整神经网络的权重和阈值,并依据程序频谱的语句数量设置虚拟单位矩阵输出每条语句的可疑值。

[0050] (6)排除偶然性正确测试用例的干扰;

[0051] 理论上只要程序执行了错误语句,程序最后的执行结果就会失败。但是在实际程序运行中并不总是如此,经常会出现执行了错误语句结果仍然为正确的情况,这样的情况被称作偶然性正确。在测试时,一次测试用例运行失败时,通常是满足以下三个条件:第一,

测试用例运行到某条错误代码行;第二,错误的程序状态被传播,并影响到后续的运行状态;第三,错误状态的传播影响到最后的输出。Masri等人通过统计分析发现了在很多的测试用例中偶然性正确是普遍存在的。

[0052] 很多时候在同一程序执行不同的测试用例程序语句会有相同的执行轨迹,但是有的测试用例执行了错误语句执行结果有的是失败的而有的却是正确。偶然性正确测试用例的存在,确实会影响程序语句怀疑值的计算。因此人为的把执行结果为正确的改为失败,或者把偶然性正确的测试用例完全去掉,排除了偶然性正确测试用例的干扰,可以提高缺陷定位的效率。

[0053] (7)依据EXAM值对程序语句进行排序,从高到低依次检查程序语句是否包含错误,直到找到错误语句。

[0054] 软件缺陷定位方法的有效性可以通过EXAM值来衡量,该值是指找到错误语句之前需要检查的语句的百分比。根据输出的语句可疑值,进行排序,依据EXAM值进行检查程序语句,直到定位到错误语句。

[0055] 图3为本发明的结果综合示例图,图中展示了传统方法和本发明提出的方法进行比较,可以看出本发明提出的方法能够检查更少的错误语句找到更多的错误版本。

[0056] 图4为本发明提出的基于次数频谱的神经网络定位方法与传统方法的比较,Tarantula、Ochiai、Dstar方法是基于频谱软件缺陷定位方法中比较有效的方法。

[0057] 图5是把偶然性正确测试用例的执行结果修改为错误;

[0058] 图6是把检查出来的偶然性正确测试用例全部删除,排除偶然性正确测试用例的干扰后,发明的方法提高了软件缺陷定位的效率。

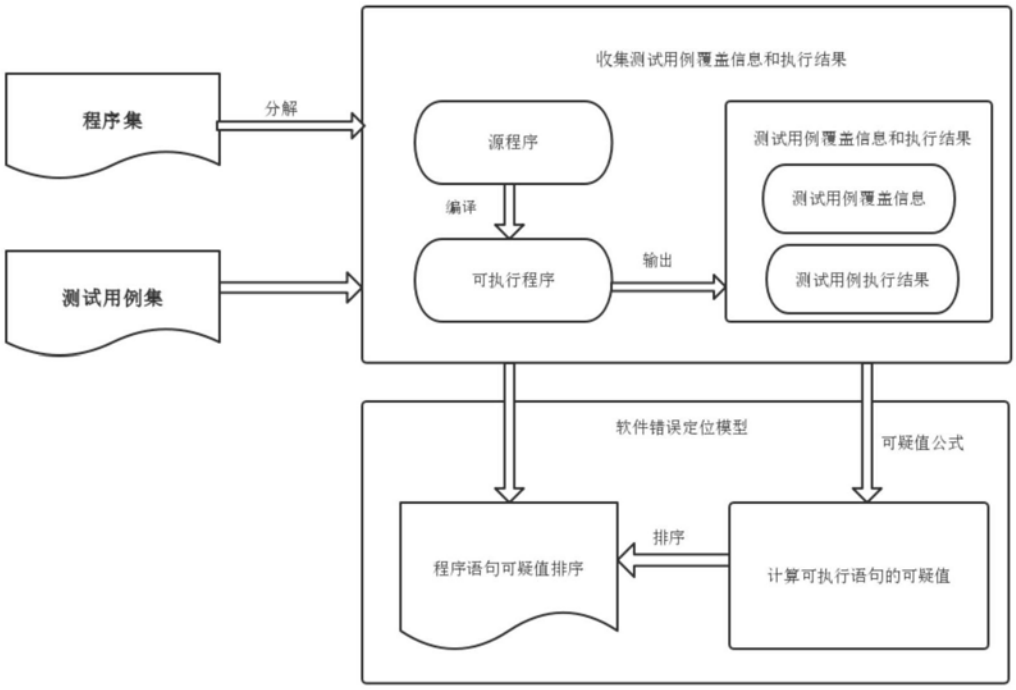

收集测试用例覆盖信息和执行结果
程序集
分解
源程序
测试用例覆盖信息和执行结果
测试用例覆盖信息
编译
测试用例集
可执行程序
输出
测试用例执行结果
软件错误定位模型
可疑值公式
排序
程序语句可疑值排序
计算可执行语句的可疑值

图1

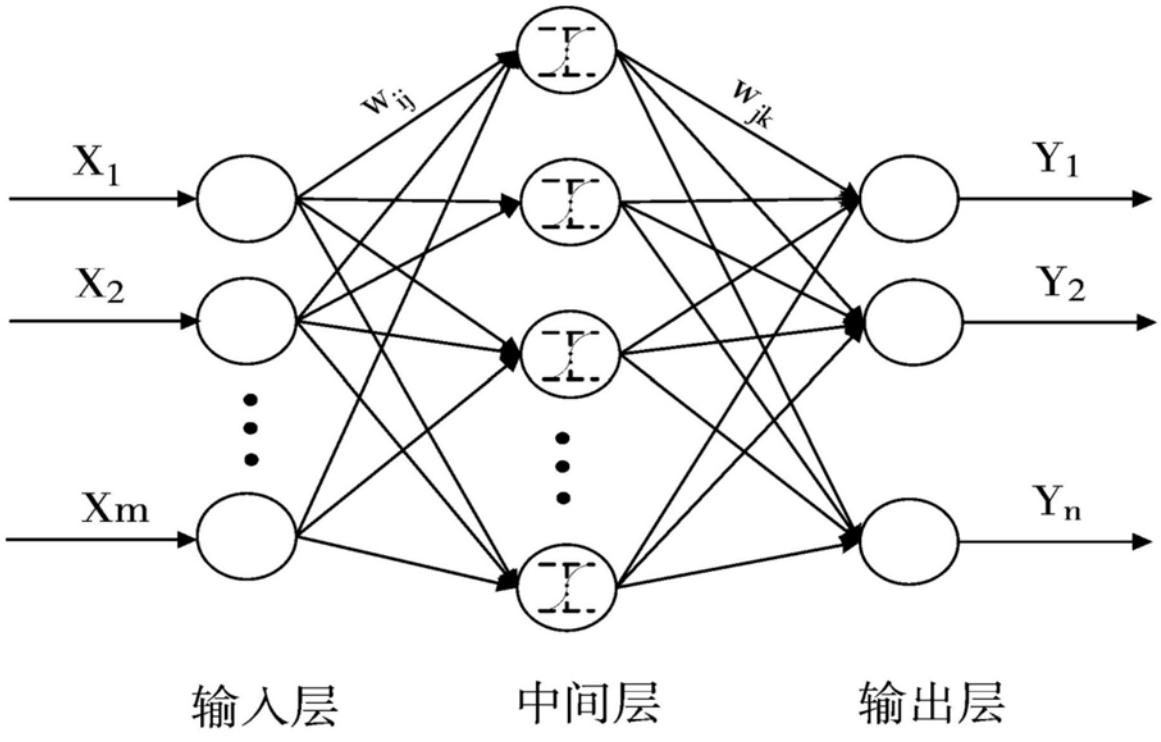

w_{ij}
w_{jk}
X_1
X_2
Xm
Y_1
Y_2
Y_n
输入层
中间层
输出层

图2

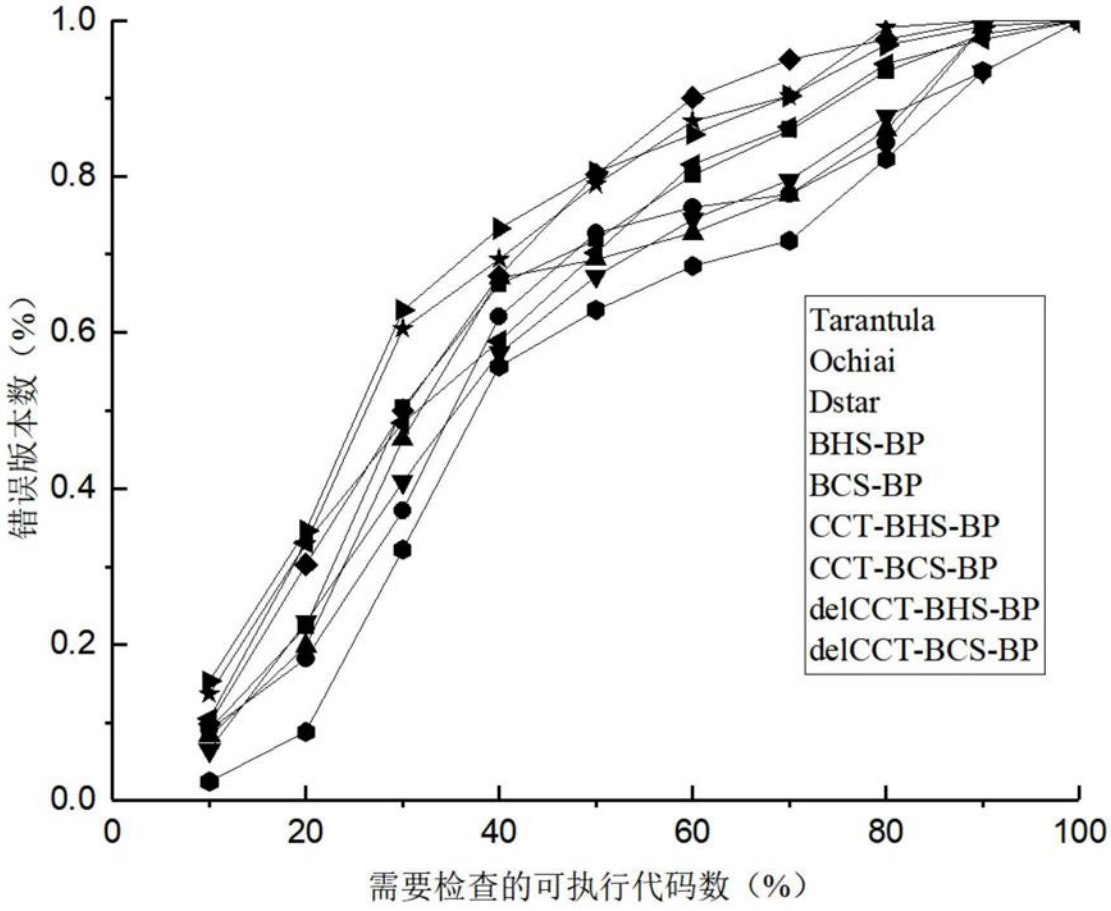

1.0
0.8
0.6
0.4
0.2
0.0
错误版本数（%）
0
20
40
60
80
100
需要检查的可执行代码数（%）
Tarantula
Ochiai
Dstar
BHS-BP
BCS-BP
CCT-BHS-BP
CCT-BCS-BP
delCCT-BHS-BP
delCCT-BCS-BP

图3

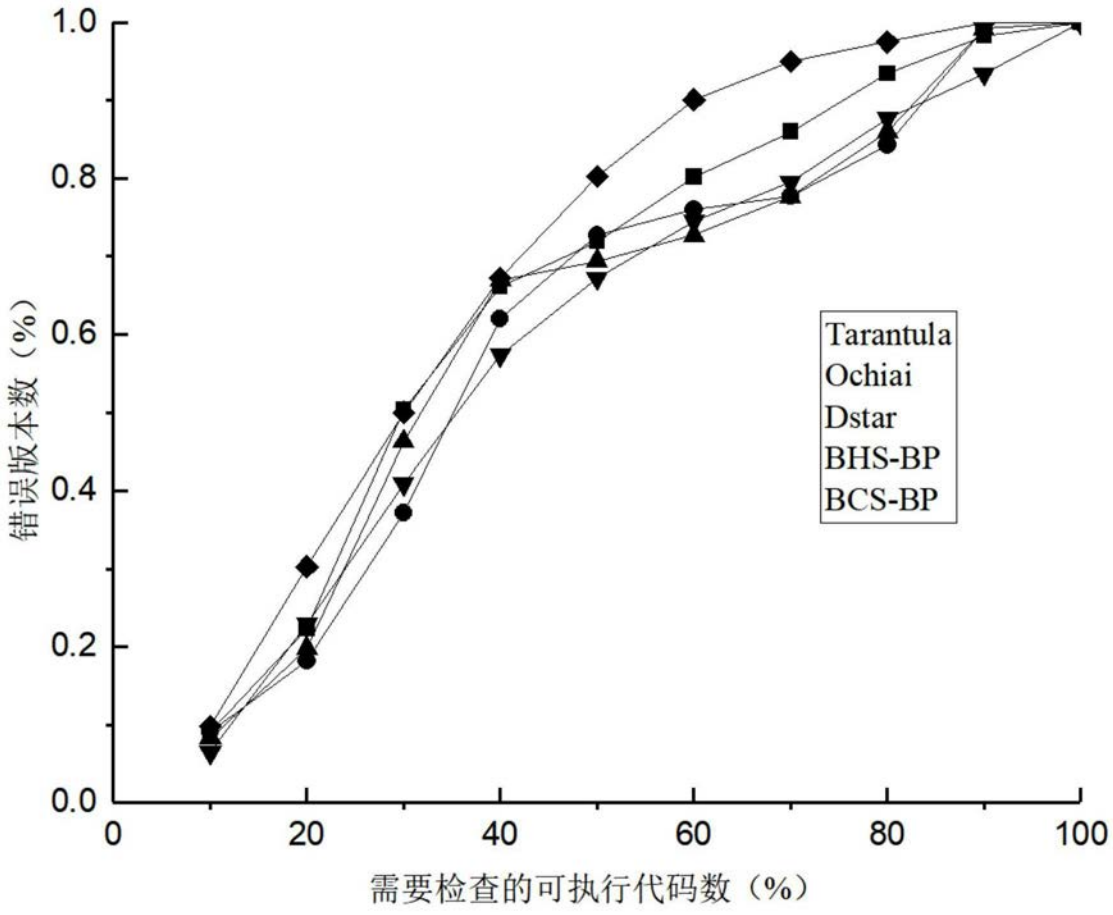

1.0
0.8
0.6
0.4
0.2
0.0
错误版本数（%）
0
20
40
60
80
100
需要检查的可执行代码数（%）
Tarantula
Ochiai
Dstar
BHS-BP
BCS-BP

图4

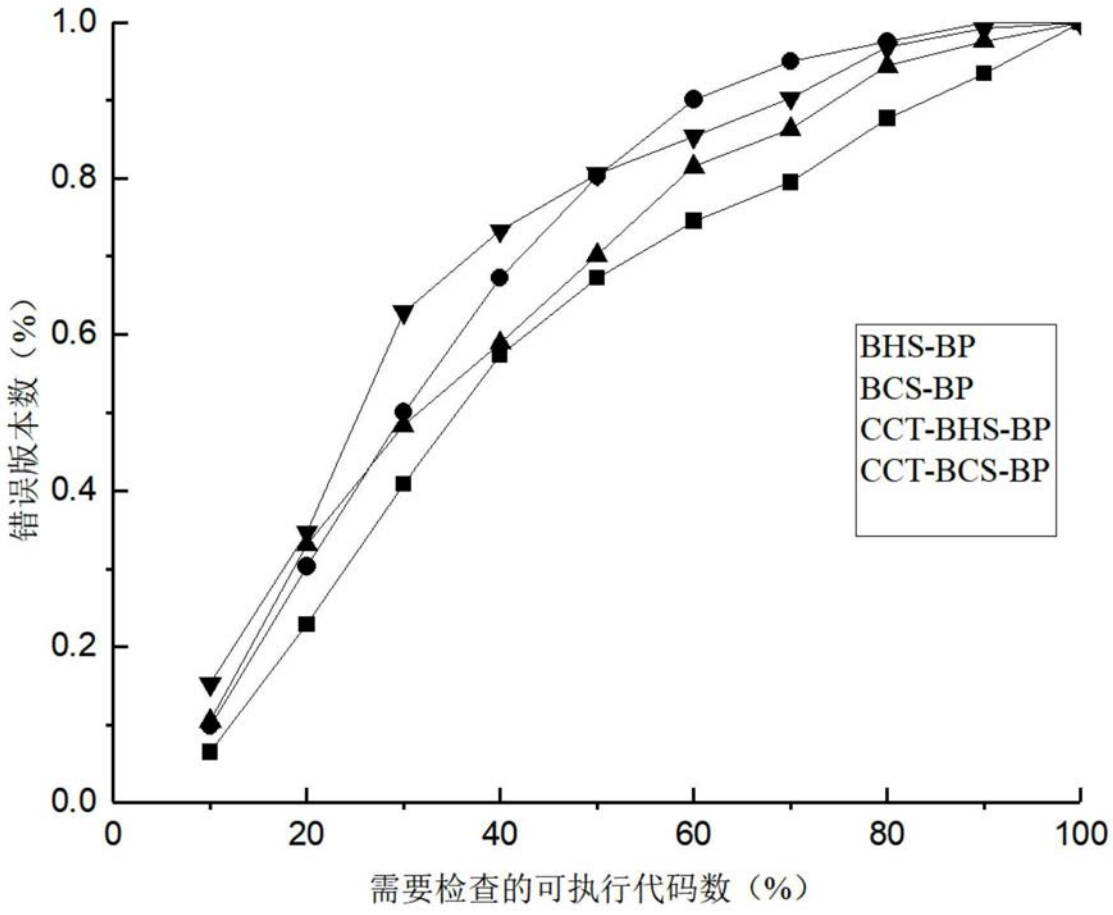

1.0
0.8
0.6
0.4
0.2
0.0
错误版本数（%）
0
20
40
60
80
100
需要检查的可执行代码数（%）
BHS-BP
BCS-BP
CCT-BHS-BP
CCT-BCS-BP

图5

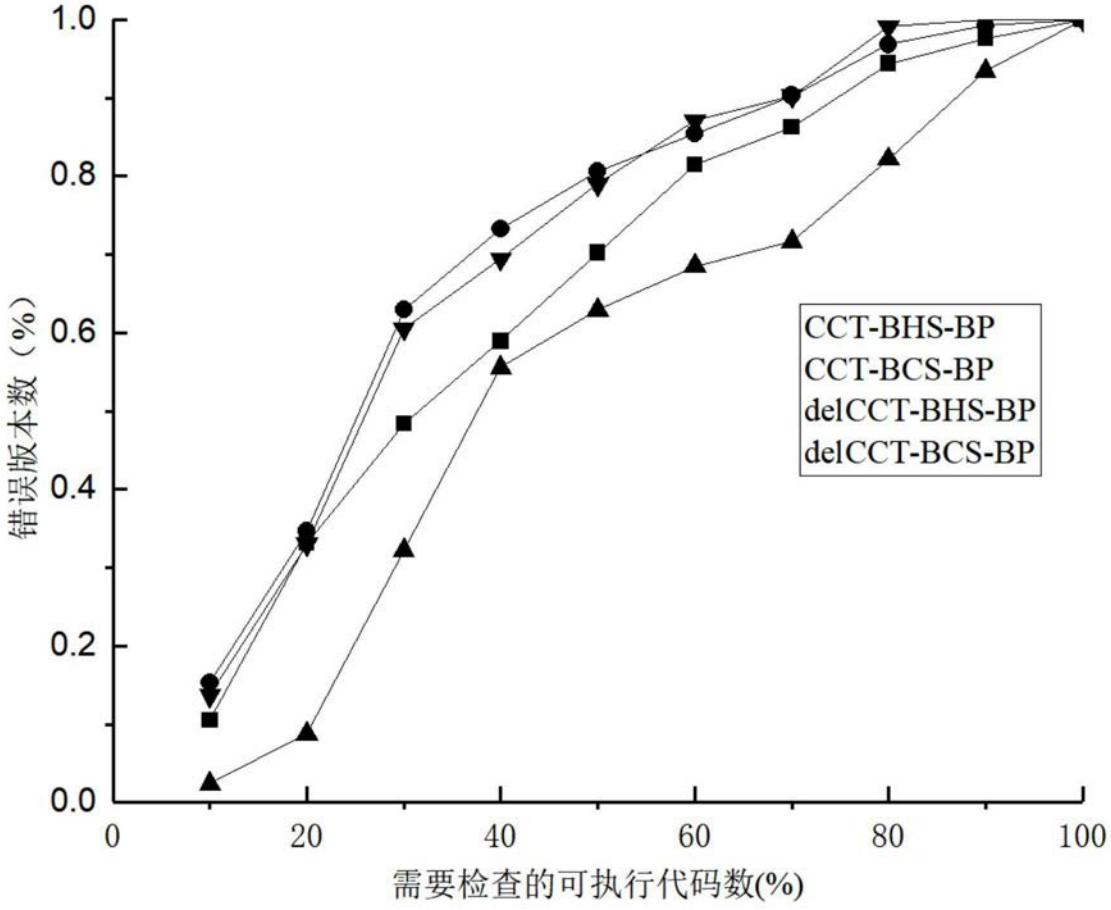

1.0
0.8
0.6
0.4
0.2
0.0
错误版本数（%）
0
20
40
60
80
100
需要检查的可执行代码数(%)
CCT-BHS-BP
CCT-BCS-BP
delCCT-BHS-BP
delCCT-BCS-BP

图6