一种基于差异分散化的错误定位方法

钱 巨[1)] 张 磊[1)] 徐宝文[2)]
[1)](南京航空航天大学计算机科学与技术学院 南京 210016)
[2)](南京大学计算机科学与技术系 南京 210093)

摘 要 错误定位技术是当前的研究热点.在各种错误定位方法中,基于最接近执行比较的方法(NN方法)[1]从成功测试执行中,选择与已发现失败执行最接近的一个,和失败执行进行比较,从而定位错误.NN方法是一种非常重要的方法,然而,实验中却发现,对于一些程序,选择最接近的成功执行与失败执行进行比较,并不能取得好的错误定位效果.为探明原因,文中首先对基于成功-失败执行比较的错误定位模型进行了研究,指出了NN方法存在上述问题的根本原因是在选择成功执行时只考虑了其与失败执行的差异数量,而忽略了差异与错误的距离这一因素.据此,提出了一种基于差异分散化的错误定位方法,其主要思想是在适当控制差异数量的同时,选择与失败执行差异最分散的成功执行,来进行错误定位.利用分散化使得部分差异能够接近错误.实验表明,该方法错误定位效果优于NN方法,性能更佳.

关键词 错误定位;定位模型;分散化;覆盖度;软件测试
中图法分类号 TP311 **DOI号** 10.11897/SP.J.1016.2015.01880

A Difference Dispersion Approach to Fault Localization

QIAN Ju[1)] ZHANG Lei[1)] XU Bao-Wen[2)]
[1)](*College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing* 210016)
[2)](*Department of Computer Science and Technology, Nanjing University, Nanjing* 210093)

Abstract Fault localization is a very active research area. In all fault localization methods, the Nearest Neighbor based (NN) method[1] selects a successful run that is closest to the failed run to compare and locate the bugs. NN method is very important and is widely used. However, we found that its fault localization effects sometimes significantly degrade even when a nearest successful run is selected. To get the reason, this paper firstly studies the basic model of the execution comparison based fault localization approaches. With the model, we found that a primary cause of NN method's degradation is that it only considers the number of differences between successful runs and failed runs while ignores the distances from the difference points to the bugs. Based on the above finding, we presented a new fault localization method based on a difference dispersion technique. Its key idea is to choose successful runs with dispersed difference points to the failed run to compare and locate the bugs. The dispersion of differences can make some difference points close to the bug and therefore reduce bug localization efforts. We conducted an experimental study on several widely used benchmark programs. The results indicate that our method is both effective and efficient.

Keywords fault localization; localization model; disperse; coverage; software test

---

收稿日期:2013-10-21;最终修改稿收到日期:2014-11-21.本课题得到国家自然科学基金(60903026)、南京航空航天大学基本科研业务费(NS2013088,NZ2013306)资助.**钱 巨**,男,1981年生,博士,副教授,中国计算机学会(CCF)会员,主要研究方向为软件分析与测试. E-mail:jqian@nuaa.edu.cn. **张 磊**,男,1987年生,硕士研究生,主要研究方向为软件分析与测试. **徐宝文**,男,1961年生,博士,教授,博士生导师,主要研究领域为程序设计语言、软件工程、并行与网络软件等.

1 引 言

质量是软件的生命，保证软件质量是一项长期而艰巨的任务.在保证软件质量的过程中，大量的资源都花费在调试上[2].调试包括定位错误、修正错误、验证修复等多个步骤，其中又以错误定位最困难、最耗资源[3].

近年来，人们对错误定位技术进行了广泛研究，提出了许多有价值的方法，包括基于程序切片的方法[4]、基于模型诊断的方法[5]、基于测试的方法[1,6-15]等.基于测试的错误定位方法效果更佳，因而是当前的研究热点.该类方法中又包括基于多个执行覆盖统计的方法[6-8]、基于多个执行谓词统计的方法[9-11]、因果链调试法[12-13]、基于最接近执行比较的方法[1,14-15]等.其中，基于最接近执行比较的错误定位方法[1](NN 方法)是一种有效且非常重要的错误定位方法.该方法的主要思想是从成功测试执行中，选择与已发现失败执行最接近的一个，和失败执行进行比较，从而定位错误.由于最终只作两个执行间的比较，这种方法不仅能够自动定位出可疑错误语句，而且也便于用户在所得结果上作进一步分析，如程序状态比较等(与之相比，多执行统计的方法所得结果涉及一大批执行，反而不便于用户在其上进行后续的人工分析).另外，由于最终只需要一个成功执行，这种方法也便于在没有成功测试的情况下，利用测试用例生成技术或程序状态改变技术，生成成功执行来进行错误定位.文献[16-21]即研究了针对错误定位的成功执行生成问题，这些研究中大多生成与失败执行最接近或非常接近的成功执行来定位错误.

尽管基于最接近执行比较的错误定位方法有着非常重要的价值，但该方法中还存在一些疑点.首先，选择与失败执行轨迹相同(绝对最接近)的成功执行进行比较时，由于程序频谱(程序执行信息的抽象，如语句覆盖、数据流覆盖等)完全相同，NN 方法无法推断任何 bug 信息，所得结果对错误定位毫无帮助.其次，在选择非常接近的成功执行进行错误定位时，尽管频谱差异可能非常小，但从这些差异去追踪 bug，有时代价却非常高，需要分析大量程序代码.上述现象表明，最接近的执行、最小的差异，并不总能保证取得最好的错误定位效果.到底什么样的成功执行，对基于成功-失败执行比较的错误定位最为有利?现有文献[1,14]等给出了直觉性的回答，但却缺少理论性的分析.

为回答上述问题，本文首先对基于成功—失败执行比较的错误定位模型进行了理论研究，总结了在该模型下，影响定位效果的两个关键因素：成功与失败执行的差异量和差异与 bug 的距离.指出 NN 方法之所以具有上述缺陷是因为只考虑了错误定位模型中的差异量因素，而没有考虑差异与 bug 的距离这一因素.仅一味追求执行最接近、差异量最少，是不全面的，很容易出现差异量少却定位效果不佳的情况.

根据对错误定位模型的分析，我们认为要获得好的错误定位效果，不宜过分追求执行最接近，而应选择成功执行，使其与失败执行的差异尽可能接近 bug，且差异量适当.过大的差异量将增加逐个分析的代价，而过小的差异量有可能得到的所有差异都偏离 bug 很远，同样将增加错误定位代价.基于上述理解，我们提出了一种新的错误定位方法，主要思想是同时兼顾影响错误定位效果的两个因素，利用差异分散化使得部分差异更容易接近 bug，利用覆盖度控制差异量，使得差异点数不至于过多或者过少.结合分散度和覆盖度选择具有适当差异数量、差异较分散的成功测试执行，与失败测试执行进行频谱比较，获得差异报告，从而定位错误.分散化的思想与自适应随机测试(ART)[22]类似，ART 用分散化的方法使得部分测试用例更容易覆盖到 bug，本文用分散的方法使得部分差异能够较接近 bug.

为检验所提出方法的有效性，本文实现了基于差异分散化的错误定位方法(DD 方法)，并将其与 NN 方法进行了实验比较.结果表明，采用 DD 方法，能够成功进行错误定位的程序版本更多，在 DD 和 NN 方法均能定位的程序版本中，约 60%的版本可以获得比 NN 方法更好的错误定位效果，即定位错误时需要检查的代码更少；DD 方法对所有程序的平均定位效果也好于 NN 方法；在定位效率方面，DD 方法由于不需要复杂的距离计算而更加高效.相对 NN 方法，平均只需要 2%～20%左右的时间，即可获得错误定位报告.总体而言，差异分散化方法无论是定位效果还是定位效率都优于基于最接近执行比较的方法，该方法是有效的.

本文第 2 节介绍研究背景和动机；第 3 节对基于成功-失败执行比较的错误定位模型进行分析；第 4 节详细介绍本文所提出的差异分散化错误定位方

法;第 5 节给出实验研究;最后,文章分析相关工作并进行总结.

2 背景与动机

基于最接近执行比较的错误定位方法在大量的测试执行中,选择与失败测试执行最接近的成功执行(差异最小),与已发现的失败执行进行频谱比较,得到差异报告,然后以此报告为起点来分析定位错误.其中,最接近执行的选择主要通过评估两个执行所对应频谱之间的相似距离实现.差异报告一般包括成功和失败执行的频谱中一系列造成不同的语句.从差异报告出发,可通过追踪程序语句间的依赖关系,定位到错误[1].具体过程如图 1 所示,该图中,F表示失败的测试执行,S_i 表示成功的测试执行.

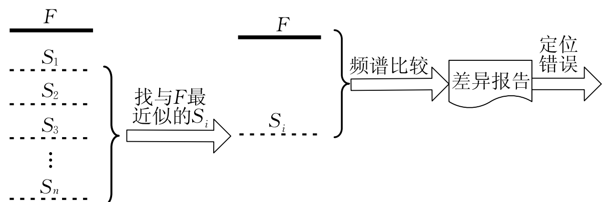


图 1 基于最接近执行比较的错误定位方法示意图

尽管 NN 方法是一种重要且应用广泛的错误定位方法,但该方法仍存在两个疑点:

(1) 在选择与失败执行轨迹完全一样(最相似)的成功测试执行进行错误定位时,由于执行比较所得的频谱差异集为空,将得到空的错误定位报告,这些报告无法用来帮助错误定位,即极端最接近的执行对错误定位完全无用.Renieris 和 Reiss[1] 的论文中发现了大量的这种情况,但作者并未对此进行深入讨论,而是在实验中直接将它们排除了.但我们认为这种排除缺乏依据,极端情况下的失效往往表明某个方法存在问题.

(2) 对一些失败的测试执行,用 NN 方法可以找到一个与它差异非常小的成功执行,然而,比较这两个执行的频谱差异去定位错误可能效果非常差,需要检查大量的代码才能找到错误.例如,对 Siemens 基准程序库[23] 中 replace_v14 程序利用 NN 方法定位错误时,找到的最接近的成功执行与失败执行差异非常小,只有一行代码的差别(第 112 行),如图 2 所示.然而,从该行开始,根据程序语句间的依赖关系去定位错误时,却发现需要检查整个程序约 85%的代码才能找到错误(bug 在第 370 行).

```
源程序 bug 行:
  369      case NCCL:
  370   if ((lin[*i]!=NEWLINE))
           /* missing code && (!locate(lin[*i],pat,j+1))) */
  371      advance=1;
  372   break;

最接近执行比较法(NN 方法)差异行:
  112 junk = addstr(src[*i],dest,j,maxset);
```

图 2 NN 方法定位效果不佳的一个案例

上述情况表明,差异大小可能并不是影响错误定位效果的唯一因素,最接近执行是否是错误定位的最佳选择仍有待进一步的讨论.本文将试着分析 NN 方法出现上述问题的原因,探索什么样的成功执行是调试的最佳选择,并在此基础上尝试对现有的错误定位方法进行改进.

3 基于成功-失败执行比较的错误定位模型

为解释 NN 方法的两个疑点,本节对基于成功-失败执行比较的错误定位模型进行了研究,并深入分析了影响错误定位效果的关键因素.

3.1 基本模型

对于基于成功-失败执行比较的错误定位方法,首先,我们认为定位错误是一个从成功、失败执行的差异现象追踪到错误的过程.即从差异现象出发,沿着程序的控制流关系、依赖关系[4] 甚至因果链[12],追踪程序执行,直到发现 bug 语句的过程.

实际调试中,差异现象可能是程序输出、程序崩溃点、断言检查失败点等,人们也正是从这些差异出发,阅读程序,分析执行(潜在地追踪控制流或依赖关系等)去定位错误.自动错误定位中,差异现象可能是程序执行中的控制差异(如语句覆盖差异),或数据差异(如程序状态差异).例如,Zeller[12] 的方法根据状态差异去定位错误,而 NN 方法则是根据控制流差异去进行定位.在没有依赖图时,可以追踪控制流图进行错误定位,而有程序依赖图时,可利用依赖图[4] 进行更高效的定位,若有因果链[12] 等,从现象追踪 bug 的过程还可以更加高效.具体错误定位过程如图 3(a)所示.

3.2 影响错误定位效果的关键因素

本文不讨论从差异现象到 bug 的追踪过程改进对错误定位的影响.这种情况下,对于只有一个差异现象的情况,影响错误定位效果的关键是差异现象与 bug 之间的距离.如图 3(b)所示,如果差异点

距离 bug 点非常远(从控制流、依赖图等上看),那么从这样的差异去定位 bug 时,需要的追踪分析量将非常大,定位效果较差.反之,如果差异点距离 bug 很接近,那么从差异现象出发,只需要很少的分析,就可以定位到 bug.当差异现象本身就发生在 bug 点上时,比如在 bug 点上崩溃,那么从差异现象可立即定位到 bug.

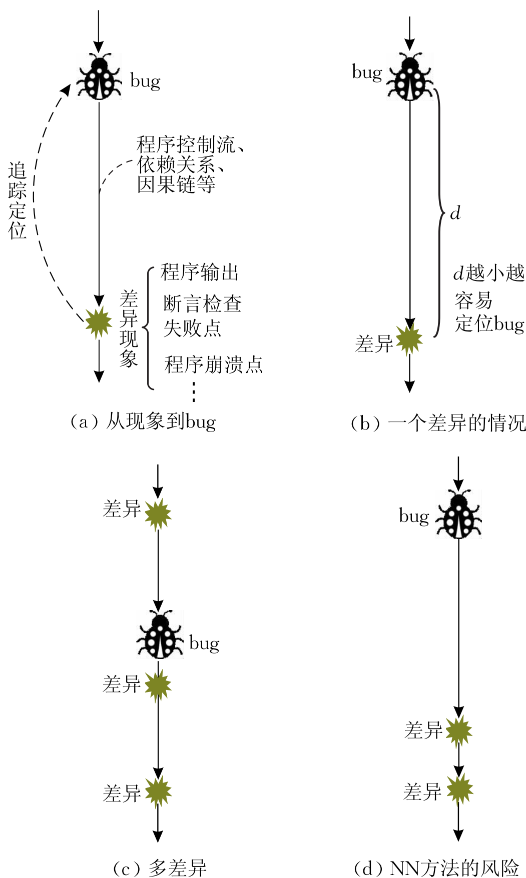


图 3　基于成功-失败执行比较的错误定位模型

对于存在多个差异的情况(如图 3(c)所示),我们认为,如果按照 Renieris 和 Reiss[1] 给出的沿着程序依赖图进行广度优先遍历,从差异现象追踪 bug 点的方法,那么定位效果至少与两个因素有关:差异现象与 bug 点的接近程度以及差异现象的数目.要取得好的定位效果,差异现象应尽量接近 bug,同时数目适当.如果每个差异都离 bug 很远,那么追踪到错误的代价一定很大;如果差异现象数目过多,会直接导致分析代价过大,而如果差异现象过少,则所有差异都离 bug 很远的风险很大,也容易造成错误定位效果不佳.

3.3　NN 方法可疑点的解释

在 NN 方法中,对于成功-失败执行的比较,有价值的只有控制差异,而没有数据差异.由于不同测试用例之间并没有建立输入数据上的关联,输入数据可能差别很大.这时,比较两个执行的数据差异并没有意义,所得差异更多地只是反映输入的差别,而不能反应 bug 的影响[24].与数据差异类似,NN 方法中控制差异尽管对错误定位有一定的价值,但也不能完全反映 bug 的影响.由于不同测试的输入并不存在相关性,控制差异点并不天然地接近 bug.比如,假设 bug 发生在程序头,而程序尾处,由于输入的某个变化,使得某一分支语句发生了转向,执行反而正确了.这时差异发生在程序尾,与 bug 距离就很远,而这种较远的距离是由输入差异造成的.

NN 方法中,只能根据控制差异去定位错误,而这些差异点本身并不天然地接近 bug.在此特点下,由本节的模型分析可知,NN 方法造成第 2 节所述问题的主要原因是只考虑了差异现象的数量,而没有考虑差异与 bug 之间的距离.对于执行轨迹相同的情况,差异量达到最小(0),这时差异与 bug 之间的距离为无穷远,因此,显然错误定位效果极差.对于差异量极小的情况,存在一种风险,即少量的几个差异现象都离 bug 非常远,如图 3(d)所示.在这种情况下,从这些差异去定位错误需要检查大量的程序代码,错误定位效果无疑也很差.

4　基于差异分散化的错误定位方法

以上对错误定位模型的分析表明,在不考虑输入的情况下,选择成功执行进行错误定位,不宜过分地追求差异点少,还应兼顾差异点与 bug 之间的距离.两者达到一定的平衡时,才可能取得最佳的错误定位效果.为此,本文借鉴 ART 测试[22] 的思想,提出了一种差异分散化的错误定位方法(DD 方法).其特点是在适当控制差异数量的情况下,利用差异点分散化,使得部分差异能够比较接近 bug,从而提高错误定位的效果.

差异分散化的主要目标是规避差异点距离 bug 太远的风险.基本思想是通过控制成功测试执行的选择,使得成功执行与失败执行的差异尽可能分散.分散度越高,差异在整个程序中分布越均匀.对于处在程序中某一固定位置的 bug 而言,越容易被部分差异点逼近(如图 3(c)所示),从而更有利于 bug 的追踪定位.分散化可以避免差异点过于集中,从而造成集体失效的情况.这一点与 ART 测试非常接近,ART 测试通过分散化测试用例,避免所有测试用例

都集中在同一个功能上反复测试,从而提高测试效率.我们通过分散化差异点,使得各个差异尽可能有不同贡献,从而提高错误定位效果.

4.1 差异分散化方法的基本步骤

本文的差异分散化方法主要包括 3 个步骤,如图 4 所示.

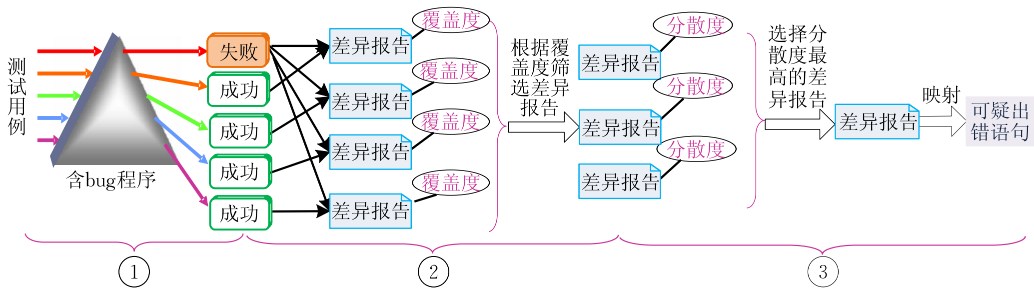


图 4 差异分散化方法的基本步骤

第 1 步(图 4 过程①),在含 bug 程序上运行测试用例,判断执行是否成功,并收集执行频谱.图 4 中,“成功”、“失败”框分别表示成功和失败执行的频谱.

第 2 步(图 4 过程②),比较失败执行和每个成功执行的频谱,得到它们的差异报告(*F*-*S*),每个报告中包含一组差异点;为这些差异报告分析差异量,计算覆盖度;然后从各个差异报告中筛选覆盖度在一定范围内的报告,以供后续处理.

第 3 步(图 4 过程③),为上一步所得的差异报告计算分散度,差异点越分散,越容易使部分差异接近 bug;根据分散度属性,选择分散度最高的差异报告,将其中的差异点映射回源代码,得到可疑出错语句.可疑出错语句将作为本方法最终所得的错误定位结果,提供给用户,以便于后续的人工分析.

4.2 差异分散度评估

类似 NN 方法,在 DD 方法中,我们用基本块覆盖作为程序的执行频谱,两个频谱之间的差异是一组基本块的集合.差异基本块按其在程序中出现的先后顺序散布在源代码中,计算相邻两个差异基本块之间间隔距离的方差,即可评估差异分散度.方差越大,表明基本块散布越不均匀,分散度越低.反之,方差越小,散布越均匀,分散度越高.

为计算相邻两个差异基本块之间的间隔距离,我们首先将基本块映射回源代码,获得基本块的行号信息,然后利用行号差值进行距离计算.每个基本块对应一个行号范围,为计算距离的方便,我们取其中心行号,即基本块首行行号加上尾行行号除以 2 的整数结果,作为基本块本身的行号(也可取首行或尾行行号,但中心行号更有代表性,能更准确地表达基本块位置).如此,每个差异基本块将对应唯一的一个行号.图 5 给出了一组差异基本块及其行号的示例.其中,$B_1,B_2,\cdots,B_n$表示一组在源代码中顺序排布的差异基本块,B_0和B_{n+1}是虚拟基本块,分别对应程序的起始和结束.引入B_0和B_{n+1}是为了计算差异分散度的方便,它们可用来计算第一个差异基本块和最后一个差异基本块到程序开头和结束的距离.图 5 中,$L_1,L_2,\cdots,L_n$是每个基本块的中心行号.L_0对应程序的起始行号,L_{n+1}对应程序的结束行号.

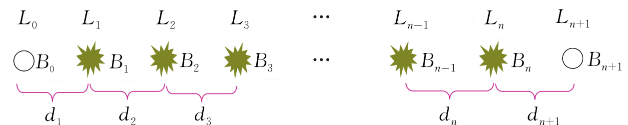


图 5 差异分散度计算

根据行号,可计算差异基本块之间的间隔距离,即间隔行数.图 5 中$d_1,d_2,\cdots,d_n,d_{n+1}$分别表示各相邻差异基本块之间的间隔距离.$d_i$的计算公式如下:

$$d_i=L_i-L_{i-1},\quad 1\le i\le n+1 \tag{1}$$

为避免空行对于分散度评估的影响,在距离计算前,我们先对程序做一个简单的处理,去除所有空行和注释行,如此每个距离即对应了各个差异基本块中心之间相隔的有效代码行数.此外,若程序存在多个文件,则这些文件将被逐个拼接起来当作单个文件进行分析.

利用差异基本块间的间隔距离即可计算各差异点之间的分散度.本文用$d_1,d_2,\cdots,d_n,d_{n+1}$这组距离之间的方差的逆作为差异分散度的评估.方差越小,分散度值越高.记 $Even(F\text{-}S)$表示失败执行 F 和成功执行 S 频谱差异的差异分散度,则其具体的评估公式如下:

$$Even(F\text{-}S)=\frac{n+1}{(d_1-\bar{d})^2+(d_2-\bar{d})^2+\cdots+(d_n-\bar{d})^2+(d_{n+1}-\bar{d})^2} \tag{2}$$

其中，$\bar{d}$ 为 $d_1, d_2, \cdots, d_n, d_{n+1}$ 距离的平均值，$Even(F\text{-}S)$值越大，差异分散度越高.

4.3 基于覆盖度和分散度的错误定位报告生成

错误定位过程中，选择不同的成功执行与已发现的失败执行进行比较，所得的差异数量可能并不相同. 有些成功执行对应的差异数大，有些差异数小. 根据第 3 节的分析，我们认为选择具有适当差异数目的成功执行，进行执行比较，是取得较好错误定位效果的一个关键. 为控制差异数目，本文定义了差异覆盖度这一指标. 利用覆盖度可以很方便地进行成功执行的选择. 差异覆盖度定义为成功和失败执行的差异基本块占整个程序基本块数目的比例，具体计算公式如下：

$$Cover(F\text{-}S) = N_D / N_P \tag{3}$$

其中，N_D表示差异中基本块的个数，N_P表示整个程序中所有基本块的个数.

为避免选择具有过多或过少差异数目的成功执行，从而直接增大错误定位代价，或通过差异到 bug 的距离这一因素间接增加错误定位代价，本文在差异覆盖度上定义了一个上下界范围，以约束成功执行的选择. 差异覆盖度范围用$[L, H]$表示，L 是所允许的覆盖度下界，H 是所允许的覆盖度上界. 上下边界主要根据经验确定，可对所有程序设置同一个上下边界，也可根据程序类型进行边界设置.

有了差异的覆盖度和分散度这两个属性，具体的从成功执行中选择某个执行生成差异报告的过程如算法 1 所示. 在该算法中，首先利用 $Cover$ 函数控制差异量，从成功测试执行集 Φ_S 中过滤掉部分差异覆盖度不在$[L, H]$范围内的执行，然后利用 $Even$ 函数选择差异最分散的一个成功执行，与失败执行进行对比，获得错误定位结果.

算法 1. 差异报告(错误定位报告)生成算法.

输入：失败测试执行 F、成功测试执行集 Φ_S，
覆盖度范围$[L, H]$

输出：作为错误定位结果的差异报告 $Report$

过程：

```
1.  Even_max := 0
2.  Report := ∅
3.  FOREACH S∈Φ_S DO
      //选择覆盖度在指定范围内的
4.    IF Cover(F-S)∈[L,H] THEN
5.      e := Even(F-S)
        //选择分散度最高的
6.      IF e > Even_max THEN
7.        Even_max := e
          //作为定位结果的差异报告
8.        Report := F-S
9.      END IF
10.   END IF
11. END FOR
```

利用算法 1，可得到一个作为错误定位结果的频谱差异报告，该差异报告包含了一组差异基本块，将这些基本块映射回源代码，可得到可疑的出错语句. 用户可以根据该可疑出错语句集的指引，沿着程序流图或程序依赖图去追踪程序执行，从而锁定最终的出错代码. 通过分散度的控制，有更大的几率使得部分差异离 bug 比较接近，这将给错误定位带来方便.

5 实验分析

为检验本文所提出方法的有效性，我们实现了一个错误定位及其效果评估工具，并在错误定位领域通用的 Siemens 基准程序库和 space 程序[23]上进行了详细的实验分析，以确定差异分散化过程中相关参数的设置，同时与基于最接近执行比较的方法(NN 方法)进行错误定位效果和效率比较.

本实验拟回答以下 3 个基本问题：

(1) 差异覆盖度范围如何确定？即覆盖度区间$[L, H]$如何设定有利于获得最佳错误定位效果.

(2) 与 NN 方法相比，DD 方法的错误定位效果如何？即 DD 方法是否定位更为准确，能够更多地排除那些原本需要检查的代码.

(3) 与 NN 方法相比，DD 方法的错误定位效率如何？即 DD 方法时间性能如何，获得错误定位报告所花费的时间比 NN 方法更多，还是更少？

5.1 实验设计

为确定何种差异覆盖度范围对错误定位最为有利，本文首先采样一批覆盖度点 $C_1, C_2, \cdots, C_n$，然后在每个覆盖度点上，计算采用差异分散化方法进行错误定位能够达到的定位效果评分 $Score_1, Score_2, \cdots, Score_n$(定位效果的量化评估将在下一段落介绍)，从而得到错误定位效果随覆盖度变化的趋势图. 在该趋势图上，找出最大定位效果对应的覆盖度范围，即可确定覆盖度区间 $[L, H]$.

为获得一个覆盖度 C_i 上的错误定位效果评估，对于单个程序版本的单个失败执行，本文在差异覆盖度固定为 C_i的层次上，筛选对应差异分散度最大的成功执行，来获得相应的定位效果评估值. 一个程序版本可能有多个失败执行，该程序版本上的错误定位效果评估值，是其所有失败执行上相应评估值

的平均.一个程序又可能有若干子版本,该程序上的错误定位效果评估值是其所有子程序版本上相应评估值的平均.总体而言,差异覆盖度 C_i 上的错误定位效果评估值 $Score_i$ 是各程序在覆盖度 C_i 上的错误定位效果评估值的平均.经过上述处理后,以差异覆盖度为 X 轴,错误定位效果评估值为 Y 轴,可得到差异分散化方法中的错误定位效果随差异覆盖度变化的趋势图.利用该图即可确定覆盖度区间 $[L,H]$.

为了评估 DD 方法的错误定位效果,与 NN 方法一样,本文采用基于依赖图的评分机制[1]来为每个错误定位报告进行打分.评分公式如下:

$$Score=1-|DS_*|/|PDG| \tag{4}$$

其中,$|PDG|$ 表示含 bug 程序的整个依赖图中的节点总数,令 $DS(n)$ 表示在依赖图中,可从错误定位报告中的节点出发,最多经 n 条有向边到达的那些节点,则 DS_* 表示使 $DS(n)$ 中包含错误语句的最小的 n 对应的可达集(如有多条错误语句,只需包含其中一条即可)[1].

式(4)中的评分表示从所得错误定位报告出发,采用广度优先搜索技术,沿着依赖图定位 bug 时,可忽略检查的代码占整个程序的比例.显然,评分越高错误定位效果越好.

上述评分机制只针对一个程序版本的一个失败执行.对于具有多个失败执行的某一程序版本 v,假定它有 m 个失败执行,分别得到 m 个错误定位报告,第 i 个报告的定位效果评分为 $Score_i$,则程序版本 v 上的错误定位效果评分 S^v 为 m 个报告评分的平均,即

$$S^v=\left(\sum_{i=1}^{m} Score_i\right)\Big/m \tag{5}$$

类似地,对于具有多个版本的某个程序 P,假定它有 n 个版本,第 i 个版本评分为 S_i^v,则该程序的错误定位效果评分 S^P 是 n 个版本定位效果评分的平均,即

$$S^P=\left(\sum_{i=1}^{n} S_i^v\right)\Big/n \tag{6}$$

最终,对于所有的 k 个程序而言,设其第 i 个程序的定位效果评分为 S_i^P,则所有程序上错误定位效果的平均评分是各个程序定位效果评分的平均,即

$$\overline{Score}=\left(\sum_{i=1}^{k} S_i^P\right)\Big/k \tag{7}$$

依据这组不同粒度的错误定位效果评分机制,可以对不同错误定位方法的定位效果进行全面评估,来考察一个方法对不同程序的效果差异,以及对所有程序的总体效果.

为比较差异分散化方法和 NN 方法的错误定位效率,本文还收集了两种方法为各个程序获得错误定位报告的平均时间,从而评判两种方法的性能.在平均时间的计算中,对一个程序 P,假定它有 n 个版本,第 i 个版本得到所有错误定位报告所需的时间为 T_i^v,则该程序上各版本得到错误定位报告所需的平均时间 T^P 为

$$T^P=\left(\sum_{i=1}^{n} T_i^v\right)\Big/n \tag{8}$$

T^P 越大表明定位错误花费的时间越多,定位的效率越低,反之则定位效率越高.

5.2 实验实现

为进行实验分析,本文基于 LLVM 编译平台①实现了一个错误定位原型工具.利用 LLVM 插桩目标程序,从而获得其执行频谱.LLVM 还提供了从基本块到源代码的映射,而源代码中的行号信息将被用来计算差异分散度.

利用所得错误定位工具,我们在 Siemens 基准库的 7 个程序和更大规模的 space 程序上进行了实验分析,并将本文所提出的方法与 NN 方法进行了对比.表 1 列出了本文的实验对象,每个实验程序有若干版本,共计 170 个程序版本,每个版本都含有一些人工植入的错误,如代码删除、代码增加或操作更改等.对于上述实验对象,我们在分析时排除了那些 NN 和 DD 方法均无法进行错误定位的失败执行和程序版本,这些执行和版本中成功执行的频谱与失败执行相同,无法进行基于频谱比较的错误定位.此外,极少量程序版本在运行时发生崩溃,这些版本的处理较为复杂,在本实验中也将它们排除了.由于崩溃型失效现象明确,本身比较容易调试,且产生崩溃的版本极少,因此,我们认为去除它们对于实验结论影响不大.经过以上选择,本实验中最后实际处理的目标程序共 155 个程序版本.

表 1 实验对象

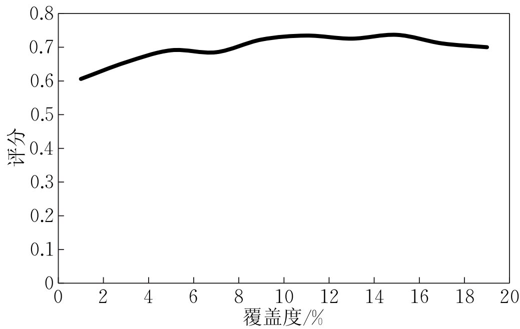
程序	功能描述	版本数	行数
print_tokens	词法分析	7	565
print_tokens2	词法分析	10	510
replace	模式替换	32	563
schedule	优先级调度器	9	412
schedule2	优先级调度器	10	307
tcas	高度区分	41	173
tot_info	信息估量	23	406
space	语言解释器	38	9564

① The LLVM Compiler Infrastructure. http://llvm.org/

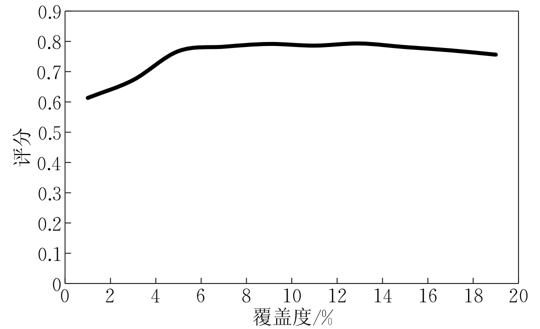
5.3　实验结果

5.3.1　覆盖度范围的选择

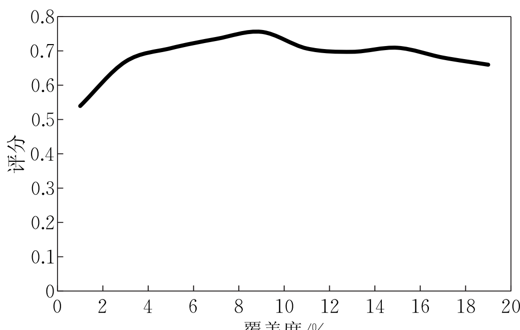
为确定覆盖度范围，根据5.1节的实验设计，我们对space以外的实验对象，进行了差异覆盖度采样分析，得到了各个实验对象上错误定位效果评分随覆盖度变化的趋势图，如图6(a)～(g)所示．在各程序错误定位效果随时间变化趋势信息的基础上，利用5.1节给出的平均化方法，可得到对于上述实验对象整体而言，错误定位效果评分的变化规律，如图6(h)所示．



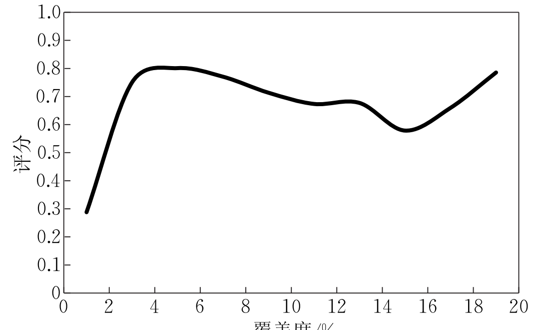
(a) print_tokens



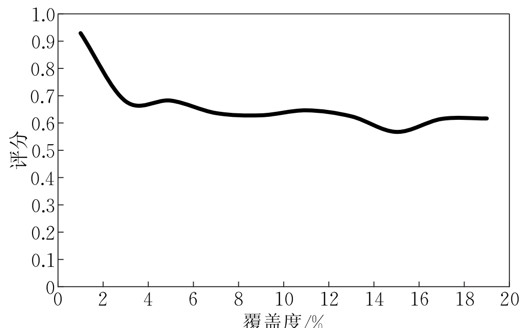
(b) print_tokens2



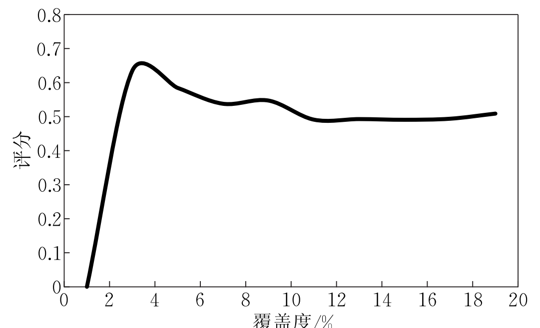
(c) replace



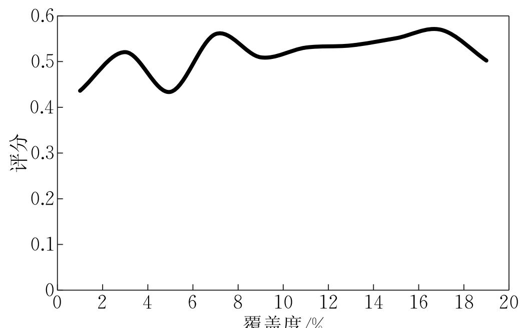
(d) schedule



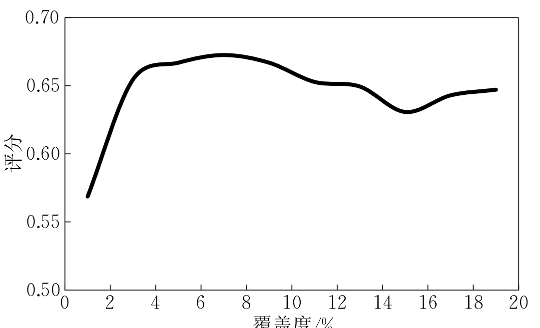
(e) schedule2



(f) tcas



(g) tot_info



(h) 各程序综合后定位效果随差异覆盖度变化的趋势图

图6　错误定位效果评分随覆盖度变化的趋势图

由图 6(a)～(g)可见，错误定位效果的一般变化趋势是：首先，在覆盖度较低时逐步上升，在中间某个较低的差异覆盖度上，错误定位效果达到最佳，然后当覆盖度再增大时，逐渐下降.即覆盖度太高错误定位效果固然不好，但也并非覆盖度越低，定位效果越好.一味追求差异少，未必能取得最佳的错误定位效果.

根据图 6(a)～(g)各子图的变化趋势以及图 6(h)的综合趋势可见，差异覆盖度在 6%～8%的范围内时，大部分程序版本都有最佳或较佳的错误定位效果，整体的错误定位效果评分也处在峰值附近.我们认为在该覆盖度范围内进行差异分散化错误定位可以取得较好的定位效果.因此，本文取差异覆盖度下界 $L=6\%$，上界 $H=8\%$，将在 6%～8%的覆盖度范围内，选择差异最分散的成功执行，进行对比，从而定位可疑出错语句.

5.3.2 错误定位效果比较

在 5.1 节所给出的评分方法的基础上，本节结合所得的具体数据，绘制了 3 个图表以比较 NN 方法和差异分散化方法的错误定位效果.

图 7 展示了所有程序版本上的错误定位效果评分情况.其中，横坐标列出了所有程序版本，纵坐标给出了各程序版本上的错误定位评分.空心柱表示 DD 方法的评分，实心柱表示 NN 方法的评分.所有程序版本按照 DD 方法评分与 NN 方法评分差值的高低进行排序，左边的版本 DD 方法效果更好，右边的版本 NN 方法效果更好.在图中，对于最左边的 18 个程序版本，NN 方法得到的差异报告为空，无法定位错误，而 DD 方法可以正常定位错误.除去这些版本，约 60%的程序版本采用 DD 方法效果更好.即对于多数程序版本，采用差异分散化方法的错误定位效果好于采用 NN 方法.

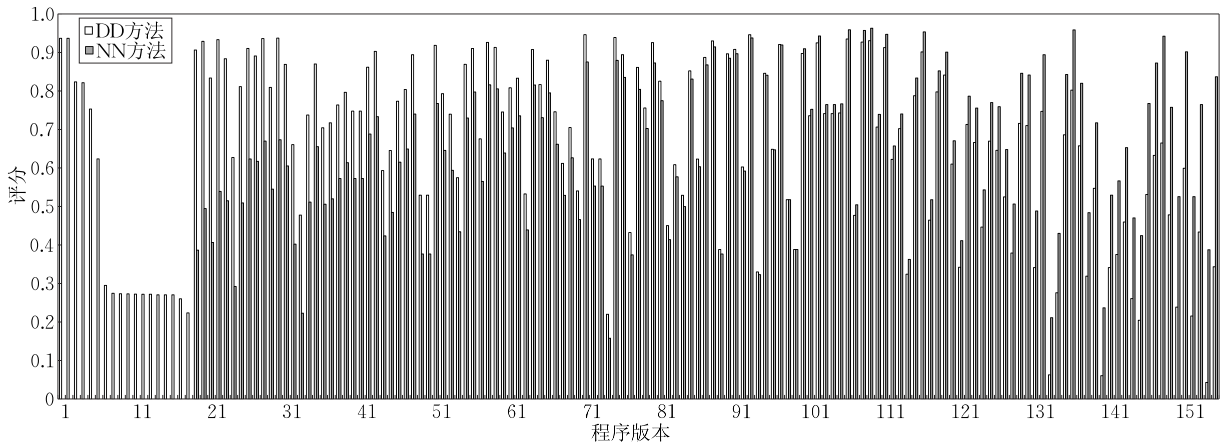


图 7 所有程序版本上的错误定位评分情况

图 8 在图 7 的基础上统计了各个评分差值级别上的程序版本数分布(只统计 DD 和 NN 方法均有效的版本).该图中横轴“评分差值”表示对于一个程序版本，用 DD 方法的错误定位效果评分减去 NN 方法的错误定位效果评分所得的结果，即两种方法的评分差值.差值以 0.1 为步长，划分为[－0.5，－0.4)，…，[－0.1，0)，[0，0.1)，…，[0.4，0.5)等多个级别.纵轴为评分差值处于某一级别内的程序版本的个数.由图 8 可知，77%的程序版本上，DD 方法要么表现更好，要么和 NN 方法评分相差在 0.1 分以内，即需要多检查的代码不超过 10%；只有 23%的程序版本上，DD 方法错误定位效果比 NN 方法弱很多.31%的程序版本上，采用 DD 方法可获得 0.1 分以上的提高，即可以少检查 10%以上的代码.另外，共 155 个版本中，有 11.6%的版本采用

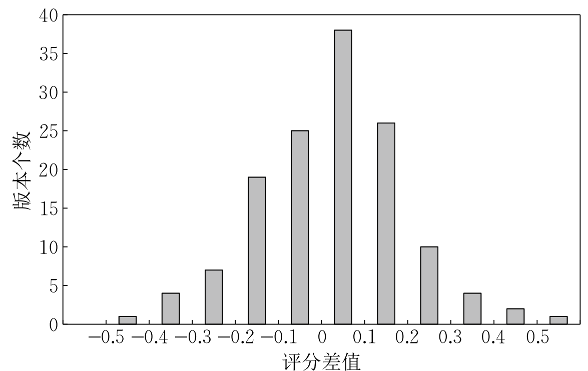


图 8 评分差异与相应程序版本个数的分布图

NN 方法无法定位错误，而采用 DD 方法可以.总体来看，对于更多的程序版本，选择差异分散化方法对获得好的错误定位效果更为有利.

图 9 以柱状图的方式给出了采用 NN 方法和

DD 方法对各个实验对象进行错误定位所得的平均评分. 由图可见，对于实验对象 print_tokens2、replace、schedule 以及 tot_info，DD 方法的定位效果评分明显好于 NN 方法；对于实验对象 print_tokens 和 space，DD 方法的评分略高；而对于实验对象 schedule2 和 tcas，DD 方法的评分则不如 NN 方法. 总体来看，采用 DD 方法，大部分程序的错误定位效果都比 NN 方法好，只有少数程序定位效果比 NN 方法差. 其中，对于 space 程序，在可以进行错误定位的 37 个版本中，DD 方法和 NN 方法各自具有优势的版本大约各占一半，平均评分 DD 方法比 NN 方法高 0.2%，尽管优势不是非常明显，但考虑到 DD 方法具有更佳的性能(如表 2 所示)，该方法仍值得采用.

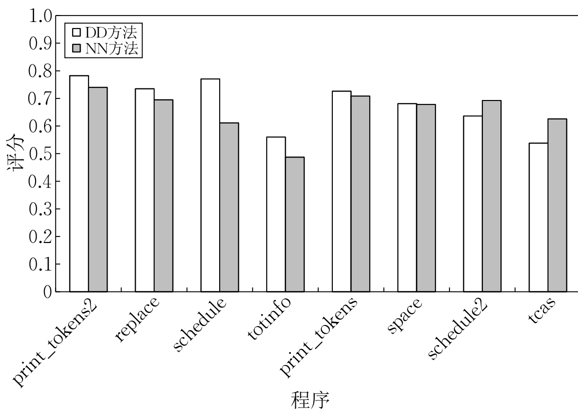


图 9 各程序错误定位效果评分柱状图

对于部分程序，差异分散化方法定位效果不如 NN 方法的主要原因是，NN 方法所得的差异点比较幸运地离 bug 很近，而该方法所得的差异点一般情况下更少，因此相应的评分也就更高. 另外，一些情况下，差异比较固定，这也削弱了筛选的作用. 事实上，DD 方法只是尽可能地规避了差异离 bug 很远的风险，因此并不总能保证差异点一定比 NN 方法离 bug 更近. 总体来看，多数情况下，DD 方法可以体现其减少差异点与 bug 距离的作用，只有少数情况下作用较弱. 我们也统计了对于所有程序的定位效果平均评分. 结果显示，NN 方法的平均评分约 65%，DD 方法的平均评分约 68%. 即就所有程序的平均情况而言，基于差异分散化的错误定位需要检查的代码更少，定位效果更好.

图 7～图 9 综合表明，较之 NN 方法，差异分散化方法能够成功进行错误定位的程序版本更多，不仅能在更多的程序版本上取得更好的错误定位效果，而且总体的平均错误定位效果也更佳.

5.3.3 定位效率比较

为评估错误定位效率，本文在 Windows Server 2008 R2 操作系统，Intel Core i5 760 CPU，16 GB RAM 的机器上收集了 DD 方法和 NN 方法为每个实验程序各版本获得错误定位报告所需的平均时间. 表 2 给出了具体的时间数据，其中的值依据 5.1 节给出的方法获得.

表 2 各程序在两种方法下获得错误定位报告的平均时间

程序	错误定位平均时间/s	
	NN 方法	DD 方法
print_tokens	6.3	0.7
print_tokens2	30.4	2.0
replace	22.8	1.3
schedule	3.0	0.3
schedule2	1.2	0.2
tcas	0.5	0.1
tot_info	1.4	0.1
space	7232.7	138.2

由表 2 可见，DD 方法进行错误定位所需的时间大约只有 NN 方法(基于 Ulam 距离)的 2%～20%左右，特别是对于 space 这样的大型程序，优势更是显著. 显然，差异分散化方法的错误定位效率比 NN 方法更高. 造成上述现象的一个重要原因是差异分散化方法只需要进行简单的方差计算，而不需要做复杂的编辑距离运算.

5.4 威胁实验有效性的因素

本实验中有一些因素可能威胁实验的有效性. 首先，尽管我们的实验代码已经仔细检查测试，但仍不排除存在错误的可能性. 代码中不当的误差传播，也可能对实验数据有一些影响. 另外，本文虽然重新实现了文献[1]中描述的 NN 方法，但由于相关实现细节并不完全清楚，因此，也不排除不同实验设置影响数据和实验结论的可能性. 实验对象中，有极少数程序版本运行时发生崩溃，本文排除了它们，这些版本也有可能影响实验结论. 再者，实验中，我们取 6%～8%作为差异覆盖度范围，该范围只是一个指导性的范围，其它范围也有可能取得更好的结论，不同类型的软件也可能需要不同的范围. 最后，本文只在 Siemens 程序集和 space 程序上进行了实验验证，相关结论是否可以推广到其它程序上，还有待更多的研究.

6 相关工作

已有的基于测试的错误定位方法主要包括基于多个执行覆盖统计的方法[6-8]、基于多个执行谓词统

计的方法[9-11]、因果链调试法[12-13]、基于最接近执行比较的方法[1,14-15]等.不同类型的方法各有特点.

其中,基于多个执行覆盖统计的方法收集成功或失败执行的代码覆盖,据此对语句进行出错可疑度排序,来定位错误.Jones 等人[6]首先根据语句在成功和失败执行中的出现次数来推定可疑度.Wong 等人[7]考虑了不同测试用例的权重,改进了定位效果.Naish 等人[8]研究了不同可疑度计算公式下的错误定位情况,为公式选择提供了指南.

与基于覆盖的错误定位不同,基于多个执行谓词统计的方法收集谓词的取值,而不是代码覆盖,来统计定位错误,其定位结果主要是与失效高度相关的谓词.Liblit 等人[9]通过对谓词和失效进行相关性分析来定位错误.Liu 等人[10]对谓词的取值规律进行分析,并将那些在失败执行中和成功执行中取值规律明显不同的谓词汇报作为可疑错误.Nainar 等人[11]改进了谓词取值的监控方法,提高了基于谓词统计的错误定位方法的效率.

上述方法在排序可疑 bug 语句方面,可以取得很好的定位效果,但它们要取得好的结果往往需要多于一个的失败执行,而且这些方法与太多的执行相关,所得的结果并不便于后续的人工分析.与之相比,基于最接近执行比较的方法只需要一个失败执行,所得的结果线索清晰,可用来和一对成功-失败执行一起进行对照分析,因此对于后续进一步的人工错误诊断和修复更为有利.我们认为,任何一个错误定位方法都不可能完美,后续的人工分析是必须的,选择一个对后续分析更有利的错误定位方法,在许多情况下有很大价值.因此,基于接近执行比较的错误定位方法仍值得进一步研究.这类方法中,目前并没有对可疑 bug 语句进行排序,未来也可类似[6-11],结合一个排序机制来进一步提高错误定位的精度.Zeller 等人[12-13]的基于因果链的调试方法采用 Delta Debugging 技术[12],修改程序状态来获得成功执行,与失败执行进行比较,从而定位导致失效的程序状态[12]及其传递过程[13].它们的方法与基于最接近执行比较的方法类似,都是比较一对执行来定位错误,前者比较程序状态,后者比较执行的代码.相比于基于状态比较的方法,基于执行代码比较的方法更易于实现,更轻量级.我们认为,这两种方法并不矛盾,未来可考虑将程序状态比较和执行代码比较结合起来,以获得更佳的错误定位效果.

在基于最接近执行比较的错误定位方面,Renieris 和 Reiss[1]的工作之后,Guo 等人[14]对文献[1]中的方法进行了改进,给出了另一种执行间距离的度量方法,新方法考虑了程序中语句的执行顺序,而不仅仅是覆盖情况.另外,Guo 等人以造成差异的分支语句作为最终的错误定位报告.尽管有所改进,但文献[14]的方法和 NN 方法本质是一样的,都以差异量最小作为选择成功执行进行错误定位的依据.该方法还存在另外一个问题,即需要收集完整的程序执行轨迹,并且对之进行复杂的对齐处理,这对于大型程序无疑代价庞大.因实现困难,我们没有与 Guo 等人的方法进行实验对比.由于本文的方法只需要收集简单的基本块覆盖信息,而不用采集程序执行轨迹,因此可以肯定的是,我们的方法比他们的方法更加高效.Guo 等人用分支语句作为错误定位报告,NN 方法和本文给出的方法也可以类似地进行改进.如此,将可以得到更好的错误定位效果.除了直接给出错误定位报告之外,Zhang 等人[15]还结合基于最接近执行的错误定位技术、动态程序切片、程序状态比较等,给出了一种断点设置方法,通过自动设置断点,可指导用户更高效地进行调试.

尽管比较失败执行和与之接近的成功执行来定位错误的方法非常有效,但有时现场并没有成功执行可用.为此,文献[16-19]提出了基于谓词替换和值替换的错误定位方法,基本思想是通过改变少量程序状态来获得与失败执行较接近的成功执行,以进行比较.除此以外,Qi 等人[20]借鉴了最近执行的思想来定位演化程序中的错误,该工作采用了符号执行技术来产生最接近的执行.Artzi 等人[21]采用了基于多个执行统计分析的方法来进行错误定位,但该工作中也借鉴了最近执行的思想,利用符号执行技术来生成与失败执行相接近的测试用例集,以便于利用优化的测试集更高效地定位错误.

7 总结与未来工作

本文从基于最接近执行比较的错误定位方法入手,分析了基于成功-失败执行比较的错误定位的基本模型,总结了该模型下影响错误定位效果的两个关键因素:差异量和差异与错误的距离.并根据该模型,提出了一种差异分散化的错误定位方法.核心思想是在适当控制差异数量的基础上,通过差异分散化,使得部分差异能够更有机会接近错误,从而减小错误定位代价.我们对所提出的方法进行了实验分析,结果表明,差异分散化方法较之基于最接近执行比较的方法,具有更好的错误定位效果,同时定位错

误花费的时间更少，定位效率更高.

在未来工作中，拟首先进一步改进本文的方法，研究更精致的差异与 bug 距离、差异量控制的方法，使之能够更准确地定位到程序错误. 另外，拟对包含数据差异的错误定位模型进行更深入分析，使得错误定位方法能够更全面地利用测试信息.

致　谢　感谢匿名审稿人的意见，它们使本文的质量有了进一步提高！

参　考　文　献

[1] Renieris M, Reiss S. Fault localization with nearest neighbor queries//Proceedings of the International Conference on Automated Software Engineering. Montreal, Canada, 2003: 30-39

[2] Collofello J S, Woodfield S N. Evaluating the effectiveness of reliability assurance techniques. Journal of Systems and Software, 1989, 9(3): 191-195

[3] Vessey I. Expertise in debugging computer programs. International Journal of Man-Machine Studies, 1985, 23(5): 459-494

[4] Xu Baowen, Qian Ju, Zhang Xiaofang, et al. A brief survey of program slicing. ACM SIGSOFT Software Engineering Notes, 2005, 30(2): 10-45

[5] Mayera W, Stumptner M. Model-based debugging—State of the art and future challenges. Electronic Notes in Theoretical Computer Science, 2007, 174(4): 61-82

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

[7] Wong W E, Debroy V, Choi B. A family of code coverage-based heuristics for effective fault localization. Journal of Systems and Software, 2010, 83(2): 188-208

[8] Naish L, Lee H J, Ramamohanarao K. A model for spectra-based software diagnosis. ACM Transactions on Software Engineering and Methodology, 2011, 20(3): 1-32

[9] Liblit B, Naik M, Zheng A, et al. Scalable statistical bug isolation//Proceedings of the ACM SIGPLAN International Conference on Programming Language Design and Implementation, Chicago, USA, 2005: 15-26

[10] Liu C, Fei L, Yan X, et al. Statistical debugging: A hypothesis testing-based approach. IEEE Transactions on Software Engineering, 2006, 32(10): 831-848

[11] Nainar P A, Liblit B. Adaptive bug isolation//Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering. Cape Town, South Africa, 2010: 255-264

[12] Zeller A. Isolating cause-effect chains from computer programs//Proceedings of the 10th ACM International Symposium the Foundations of Software Engineering. Santa Fe, USA, 2002: 1-10

[13] Cleve H, Zeller A. Locating causes of program failures//Proceedings of the International Conference on Software Engineering, St. Louis, USA, 2005: 342-351

[14] Guo L, Roychoudhury A, Wang T. Accurately choosing execution runs for software fault localization//Proceedings of the 15th International Conference on Compiler Construction. Vienna, Austria, 2006: 80-95

[15] Zhang Cheng, Yan Dacong, Zhao Jianjun, et al. BPGen: An automated breakpoint generator for debugging//Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering(Vol. 2). Cape Town, South Africa, 2010: 271-274

[16] Wang T, Roychoudhury A. Automated path generation for software fault localization//Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering. Essen, Germany, 2005: 347-351

[17] Zhang X, Gupta N, Gupta R. Locating faults through automated predicate switching//Proceedings of the 28th International Conference on Software Engineering. Shanghai, China, 2006: 272-281

[18] Jeffrey D, Gupta N, Gupta R. Fault localization using value replacement//Proceedings of the 2008 International Symposium on Software Testing and Analysis. Seattle, USA, 2008: 167-178

[19] Liu Yongmei, Li Bing. Automated program debugging via multiple predicate switching//Proceedings of the 24th AAAI Conference on Artificial Intelligence. Atlanta, USA, 2010: 327-332

[20] Qi D, Roychoudhury A, Liang Z, Vaswani K. Darwin: An approach for debugging evolving programs//Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering. Amsterdam, the Netherlands, 2009: 33-42

[21] Artzi S, Dolby J, Tip F, Pistoia M. Directed test generation for effective fault localization//Proceedings of the 19th International Symposium on Software Testing and Analysis. Trento, Italy, 2010: 49-60

[22] Chen T Y, Kuo F-C, Merkel R G, Tse T H. Adaptive random testing: The ART of test case diversity. Journal of Systems and Software, 2010, 83(1): 60-66

[23] Do H, Elbaum S G, Rothermel G. Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. Empirical Software Engineering: An International Journal, 2005, 10(4): 405-435

[24] Sumner W N, Zhang X. Algorithms for automatically computing the causal paths of failures//Proceedings of the International Conference on Fundamental Approaches to Software Engineering. LNCS 5503. York, UK, 2009: 355-369

QIAN Ju, born in 1981, Ph. D., associate professor. His research interests focus on software analysis and testing.

ZHANG Lei, born in 1987, M. S. candidate. His research interests focus on software analysis and testing.

XU Bao-Wen, born in 1961, Ph. D., professor, Ph. D. supervisor. His research interests include programming languages, software engineering, parallel and network computing, and etc.

Background

The paper concerns on the fault localization problem in software engineering. Fault localization aims to reduce the scope of codes that needs to be investigated in order to find and fix the bugs. As it can largely improve the efficiency of software debugging, lots of efforts have been devoted to this area. Many different fault localization approaches have been proposed, including the program slicing based approaches, the model-based diagnosing approaches, the testing-based approaches, and etc. We also have proposed several program slicing based approaches and testing-based approaches in our previous work. Although in the existing researches, many interesting results are found, the problem is far from completely solved. Recently, experimental studies show that the testing-based approaches are particularly effective for fault localization. The approaches exploit the execution information of many different tests and therefore can more precisely locate the faults. In the testing-based approaches, the nearest neighbor based (NN) method selects a successful run that is closest to the failed run to compare and locate the bugs. NN method is very effective and is widely used. However, we found that its fault localization effects sometimes significantly degrade even when a nearest successful run is selected. To overcome the problem, we firstly investigated the reason and found that a primary cause of NN method's degradation is that it only considers the number of differences between successful runs and failed runs while ignores the distances from the difference points to the bugs. With this finding, the paper presents a new fault localization method based on a difference dispersion technique. The key idea of the new method is to choose successful runs with dispersed difference points to the failed run to compare and locate the bugs. The dispersion of differences can make some difference points close to the bug and therefore reduce bug localization efforts. Our experimental results show that the proposed method is both more effective and more efficient compared to the NN method.

This work is partly supported by the National Natural Science Foundation of China (60903026) and the NUAA Fundamental Research Funds (Nos. NS2013088, NZ2013306). These projects study the techniques that can be used to improve the efficiency of software testing, debugging, and evolution. The approach proposed in this paper focuses on the software debugging problem. By localizing the faults more precisely, the efforts needed in debugging can be reduced.