基于动态相似度的错误定位优先排序方法

蒲进兴 李登辉 李 征 赵瑞莲
(北京化工大学计算机系 北京 100029)

摘 要 在软件测试中,错误定位优先排序通过优化测试用例的执行次序来提高错误定位的效果,并将检测错误和定位错误相结合,以降低测试成本。提出了一种基于动态相似度的错误定位优先排序方法,在相似度计算中,引入了语句怀疑度,提高了相似度计算的有效性以及错误定位的准确度;同时分析并验证了不同测试用例优先排序算法对后续定位错误的影响。在6个C基准程序上,针对3种广泛采用的测试用例优先排序算法和2种错误定位技术进行了实验,结果表明提出的方法能提高错误定位的准确度和效率。

关键词 错误定位优先排序,错误定位,动态相似度
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2016.3.038

Dynamic Similarity Based Fault Localization Prioritization

PU Jin-xing LI Deng-hui LI Zheng ZHAO Rui-lian
(Department of Computer Science, Beijing University of Chemical Technology, Beijing 100029, China)

Abstract Fault localization prioritization (FLP) reorders test cases to improve efficiency of fault localization. It combines the two key processes in software regression testing, fault detection and fault localization, in order to reduce the test cost. We proposed a dynamic similarity based fault localization prioritization, in which the statement suspicious value is introduced as a weight of similarity for test case implementation to improve the effectiveness of fault localization. The impact of different test case prioritization algorithms was empirically analyzed and validated. In the experiments, three widely used test case prioritization algorithms were used to combine with two fault localization technologies. The results based on six benchmark C programs show that the proposed approach can effectively improve the accuracy and efficiency of fault localization.

Keywords Fault localization prioritization, Fault localization, Dynamic similarity

1 引言

错误定位优先排序(Fault Localization Prioritization, FLP)是一种以定位错误为目标,把检测错误和定位错误两个过程相结合的测试用例优先排序技术[1]。FLP执行过程主要分为两个阶段,第一阶段以检测错误为目标,对测试用例进行排序并执行;当发现第一个未通过的测试用例(也称为failed测试用例,即测试用例的实际输出结果与预期结果不一致)时,转到第二阶段,以定位错误为目标,调整余下测试用例的执行顺序以提高错误定位效果。错误定位优先排序能够较早地检测到并定位出错误的位置,节约回归测试时间。

传统的测试用例优先排序(Test Case Prioritization, TCP)[2,3]和错误定位(Fault Localization, FL)[4,5]是两个独立的过程。Bo Jiang等人最早研究和分析了测试用例优先排序技术对错误定位效果的影响[6,7],发现在一定情况下,测试用例优先排序技术可以改善错误定位的效果。Alberto等人在2011年提出了基于信息熵和贝叶斯条件概率的以及基于模糊分组的测试用例优先排序的错误定位方法[8,9]。Yoo等人在2013年正式提出了错误定位优先排序方法[1],实现了发现错误和定位错误的结合,并达到利用测试用例优先排序技术来改善错误定位的效果。

错误定位优先排序方法的主要步骤包括:(1)以检测错误为目标,对测试用例进行优先排序并执行,直到第一次发现failed测试用例;(2)以定位错误为目标,按照相应排序准则,调整后续测试用例的执行序列;(3)按序执行,直到满足执行终止条件就停止;(4)根据传统的错误定位技术,利用怀疑度公式(如Tarantula[10])计算语句怀疑度值并排序,得到语句的Rank值序列,再按Rank值递减顺序,检查程序语句。在发现包含错误的语句时,检查的语句越少,说明错误定位越准确。需要注意的是,FLP是一种包含执行终止条件的基于测试用例选择的错误定位技术,并且选择过程与测试用例执行顺序有关,所以它同时是一种优先排序技术。

到稿日期:2015-02-08 返修日期:2015-05-26 本文受国家自然科学基金(61170082,61073035),教育部新世纪优秀人才计划(NCET-12-0757),留学回国人员科研启动基金(LXJJ201303)资助。

蒲进兴(1987-),男,硕士生,主要研究方向为错误定位,E-mail:475168612@qq.com;**李登辉**(1990-),男,硕士,主要研究方向为错误定位,E-mail:852522780@qq.com;**李 征**(1974-),男,博士,教授,CCF高级会员,主要研究方向为软件分析及测试,E-mail:lizheng@mail.buct.edu.cn;**赵瑞莲**(1964-),女,博士,教授,CCF会员,主要研究方向为软件测试与软件可靠性分析,E-mail:rlzhao@mail.buct.edu.cn。

目前以定位错误为目标的优先排序是一种根据测试用例覆盖信息相似度进行排序的方法，即优先执行与第一个 failed 测试用例相似度较高的测试用例[1]。表 1 给出了一个被测程序的测试用例覆盖信息和错误定位的实例。该程序包含 8 条语句以及 4 个测试用例，其中 s_7 是含有错误的语句，另外还给出了相应测试用例的执行结果(F 表示 failed，T 表示 passed)以及测试用例以不同序列执行时根据 *Tarantula* 怀疑度计算公式得到的每条语句的怀疑度值。如表 1 所列，当 t_1 执行结果为 F 时，根据目前的相似度排序方法，计算余下测试用例 t_2、t_3、t_4 与 t_1 的相似度，其值分别是 0.61、0.73、0.61，可得其优先执行顺序为 $t_3 \rightarrow t_2 \rightarrow t_4$。当执行 $t_3 \rightarrow t_2$ 时，s_6 和 s_7 两条语句的怀疑度值都为 1，而如果按 $t_3 \rightarrow t_4$ 执行，则只有包含错误的语句 s_7 的怀疑度值为 1。可以看出，当前的方法并不能发现最优的执行序列。

表 1　基于覆盖相似度的错误定位优先排序实例

语句	测试用例				Tarantula 怀疑度			
	t1	t2	t3	t4	t1	t1→t3	t1→t3→t2	t1→t3→t4
s0	●	●	●	●	1	0.5	0.5	0.5
s1	●	●	●		1	0.5	0.5	0.67
s2	●	●	●	●	1	0.5	0.5	0.5
s3	●		●		1	0.5	0.67	0.67
s4			●	●	0	0	0	0
s5		●			0	0	0	0
s6	●			●	1	1	1	0.67
s7 (fault)	●				1	1	1	1
执行结果	F	T	T	T				

现有测试用例相似度是指两个测试用例所覆盖语句数目的相近程度[1]，并未考虑具体语句的不同，因而不能有效区分测试用例的优先次序，即使考虑具体语句不同时，也假设每条语句所占的权重是相同的，即权重为 1，不能有效区分语句之间的差别。实际测试中，随着测试用例的执行，不同语句对错误定位的权重并不一样，语句的怀疑度值也在动态变化，同时权重随之变化，也说明语句的权重并不相同，而且语句的怀疑度值的大小能体现其权重的变化，因此本文把语句的怀疑度作为权值引入到相似度计算中，以提高相似度计算的有效性。

基于以上原因，本文针对错误定位优先排序中的测试用例相似度计算问题，提出了一种基于动态相似度的错误定位优先排序方法。首先引入语句怀疑度值作为权值加入到相似度计算中，以体现怀疑度不同的语句在错误定位中的不同权重；其次考虑到语句怀疑度值随着下一个测试用例的执行而变化，后续测试用例的排序也会相应调整，所以排序是一个动态的过程；执行过程中，通过排除准则来约减执行的测试用例数。此外，以检测错误为目标的不同测试用例优先排序算法对检测到的第一个 failed 测试用例会产生影响，而第一个 failed 测试用例又会直接影响到后续测试用例的选择。因此本文结合了 3 种不同的测试用例优先排序算法，对其错误定位效果进行了对比。实验表明，本文的方法能够提高错误定位的准确度，减少错误定位的开销。

本文主要贡献包括：(1)提出了一种基于动态相似度的错误定位优先排序方法，在相似度计算过程中引入语句怀疑度作为权值来改进相似度计算，提高了计算的有效性；(2)分析了采用不同测试用例优先排序算法检测错误时对后续错误定位效果的影响；(3)通过实验验证了本文提出的方法可以提高错误定位的准确度，并约减执行的测试用例数。

2　技术背景

本节主要介绍测试用例优先排序技术、基于语句覆盖的错误定位技术和错误定位优先排序技术等相关背景。

2.1　测试用例优先排序技术

测试用例优先排序技术(TCP)是一种根据不同的排序准则，优化测试用例的执行顺序，从而较快地发现错误，以降低回归测试开销的技术[2]。主要的测试用例优先排序算法有以下 3 种。

随机测试用例优先排序(Random Test Case Prioritization，RND)是一种最直接的测试用例优先排序方法，测试用例的执行顺序随机生成[2,11]。通常把 RND 作为其它测试用例优先排序方法性能优劣的比较基准。

额外贪心测试用例优先排序(Additional Greedy Test Case Prioritization，ADD)是贪心算法中的一种，在每次测试用例执行之前，以覆盖最多未被覆盖的语句数为目标来选择下一个测试用例[12]。

自适应随机测试用例优先排序(Adaptive Random Test Case Prioritization，ART)是一种混合型的测试用例排序方法，主要由两部分组成：(1)随机生成一个测试用例候选子集；(2)度量该子集内的测试用例与已经选取的测试用例集中所有测试用例的距离，然后通过特定距离标准选择下一个测试用例[13]，通常有 3 个距离标准：maximize the minimum、average和 maximum，在使用时一般根据需求选取合适的标准。自适应随机测试用例优先排序是一种较为优越且花销较少的测试用例优先排序方法。

2.2　基于语句覆盖的错误定位技术

基于语句覆盖的错误定位技术利用程序谱记录程序测试用例的执行信息，统计程序语句被不同测试用例覆盖的情况，再根据怀疑度计算公式度量语句包含错误的可能性，最后得到一个语句可能包含错误的概率值序列，即怀疑度值序列。在此基础上，测试人员根据该序列对程序源代码进行检查，可以有效减少定位错误时检查源代码的数量，从而降低定位错误的代价。错误定位的怀疑度计算公式有很多种，本文使用 *Tarantula*[10] 和 *Ochiai*[14] 两种：

$$Tarantula(s)=\frac{\frac{failed(s)}{totalfailed}}{\frac{failed(s)}{totalfailed}+\frac{passed(s)}{totalpassed}} \quad (1)$$

$$Ochiai(s)=\frac{failed(s)}{\sqrt{totalfailed*(failed(s)+passed(s))}} \quad (2)$$

其中，$failed(s)$ 表示覆盖语句 s 且执行结果为 failed 的测试用例数，$totalfailed$ 表示测试用例中执行结果为 failed 的测试用例总数。同理，$passed(s)$ 表示覆盖语句 s 且执行结果为 $passed$ 的测试用例数，$totalpassed$ 为测试用例中执行结果为 $passed$ 的测试用例总数。

2.3　错误定位优先排序技术

错误定位优先排序(FLP)是指在回归测试中，第一次检测到错误后，针对错误定位，调整余下测试用例的执行顺序，以尽可能地改善错误定位效果的技术[1]。此时，测试用例优先排序的目标不再是检测错误，而是更准确地定位错误。

传统上的测试用例优先排序和错误定位是两个独立的过程，前者是为了检测错误，后者是为了准确定位错误发生的位

置。但这两种技术都可依赖于同一信息，即程序测试用例的执行覆盖信息。Yoo 等人提出了错误定位优先排序，并提出了两种排序方法：(1)利用信息论和基于覆盖的错误定位技术相结合的方法，假设下一个执行的测试用例在 passed 或 failed 条件下分别计算每条语句可能包含错误的条件概率和每个测试用例的信息熵，优先执行信息熵较小的测试用例，以改善错误定位效果；(2)基于测试用例覆盖信息相似度排序的方法，当发现第一个 failed 测试用例后，选择优先执行与该 failed 测试用例相似度较高的测试用例，以提高错误定位的效果[1]。方法(1)在计算测试用例的信息熵时非常耗时，方法(2)则未考虑语句之间的差别，且一次性把所有测试用例进行优先排序，然后按序列执行，并不能发现最优的执行序列。为此，本文提出了一种基于动态相似度的错误定位优先排序方法，在每个测试用例执行后，将怀疑度变化作为反馈，动态调整测试用例的执行序列。

3 基于动态相似度的错误定位优先排序方法

本节主要讨论基于动态相似度的错误定位优先排序方法的框架及具体步骤。

3.1 方法框架

本文方法以测试用例相似度为指导来选择下一个优先执行的测试用例，在计算测试用例相似度时，引入了语句的怀疑度值，同时考虑了覆盖语句数目和语句之间的差别，在此基础上提出了基于动态相似度的错误定位优先排序方法。

图 1 为基于动态相似度的错误定位优先排序方法的框架，其主要步骤为：(1)执行测试用例，当发现第一个 failed 测试用例时，计算语句当前的怀疑度值；(2)根据语句怀疑度值，更新 failed 测试用例的特征向量(P 向量)；(3)根据相似度计算公式，计算余下测试用例与 P 向量之间的相似度；(4)选择相似度最大的测试用例优先执行，同时按照一定准则排除一些测试用例；(5)若满足终止条件，则跳出循环进行错误定位，否则返回(2)。图 1 中白色椭圆框部分是第一阶段的以检测错误为目标的测试用例优先排序，浅灰色部分是第二阶段以定位错误为目标的本文提出的动态相似度排序策略，也是本文方法的核心环节，即发现第一个 failed 测试用例后如何动态调整余下测试用例的执行顺序。

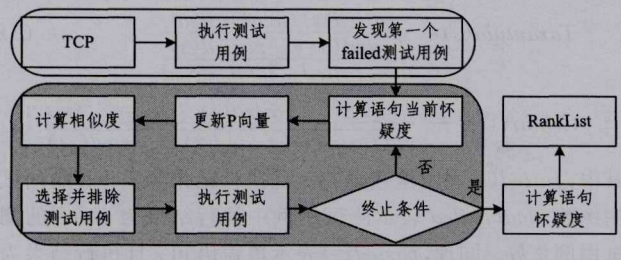


图 1 基于动态相似度的错误定位优先排序方法框架

3.2 动态相似度排序

本文提出的动态相似度排序(Dynamic Similarity Sort, DSS)是指通过计算余下测试用例与 failed 测试用例的特征向量(P 向量)之间的相似度，优先执行相似度较大的测试用例。每执行一个测试用例则同时更新 P 向量的值，以提高相似度计算的有效性，从而提高错误定位的准确度。同时按照一定准则排除测试用例，约减测试用例数目，减少错误定位的代价。

定义 1 failed 测试用例的特征向量是检测到第一个 failed 测试用例之后，执行第 i 个测试用例时，由语句怀疑度值组成的 n 维向量，标为 $P_i=(p_1,p_2,\cdots,p_j,\cdots,p_n)$，简称 P 向量；其中 n 是程序的可执行语句数，p_j 的计算方式如式(3)所示。当语句 s_j 被当前所有执行结果为 failed 的测试用例所覆盖且怀疑度值不小于 *threshold* 时，对应 P 向量中的值 p_j 为 s_j 当前的怀疑度值；否则 p_j 为 0。

$$p_j=\begin{cases}Suspiciousness(s_j,t_i,E_k), & \text{if } s_j\in S_f \text{ and } p_j\geqslant threshold\\ 0, & \text{else}\end{cases} \tag{3}$$

其中，$Suspiciousness(s_j,t_i,E_k)$ 表示在执行测试用例 t_i 时，利用怀疑度公式 E_k 计算得到的语句 s_j 的怀疑度值，$S_f=\{s|s\in(St_{f1}\cap St_{f2}\cap\cdots\cap St_{fi}\cdots)\}$，其中 St_{fi} 表示被第 i 个执行结果为 failed 的测试用例覆盖的语句集。*threshold* 是一个针对语句怀疑度值而设置的、用以排除一些怀疑度较低且不包含错误的语句的预设值。因此在计算相似度和怀疑度时，只保留那些被 failed 测试用例覆盖且其怀疑度值大于 *threshold* 的语句，因为这些语句更可能包含错误，可以降低计算成本，缩小查找包含错误语句的范围，从而提高错误定位的准确度。

DSS 在首次检测到错误时，计算语句怀疑度，此时 P 向量中对应 failed 的测试用例覆盖语句的怀疑度值都为 1，其他为 0，然后依次计算余下测试用例与当前 P 向量的相似度距离。优先执行相似度较大的测试用例，若执行结果为 failed，则对 failed 测试用例覆盖的语句取交集，即 S_f。然后每执行一个测试用例就更新一次 P 向量的值，同时按照一定的准则约减测试用例，相应准则如下。

准则一：排除相似度值相同的测试用例。相似度值相同的测试用例主要有两种情况：(1)覆盖信息完全相同的测试用例。因为其覆盖了相同的语句，执行后会使其覆盖的语句的怀疑度值同时升高或者降低，这样的测试用例对将包含错误的语句从正确语句中区分出来，帮助并不明显。(2)相似度值相等但覆盖信息不完全相同的测试用例。这样的测试用例有可能对错误定位有所帮助，但因后续执行了不同相似度值的测试用例，从而能达到相同的效果。

准则二：排除仅覆盖怀疑度小于 *threshold* 值的语句的测试用例。这些测试用例覆盖的语句怀疑度值非常低，可认为其不可能包含错误，因此可以不考虑仅覆盖这些语句的测试用例。

可以看出，本文选择优先执行相似但不相同的测试用例，且同一相似度值的测试用例仅执行一个，避免了重复执行相同覆盖或者仅覆盖较低怀疑度值语句的测试用例。

DSS 排序策略的终止条件是测试用例集中不再有满足条件的测试用例。通过以上准则可以较快达到终止条件，从而约减执行的测试用例数目，节约测试成本。

DSS 中的相似度是指计算测试用例 t_i 的覆盖向量与 P_i 向量之间的相似度。其中，一个覆盖向量即为一个测试用例的覆盖信息，由 0 和 1 组成。本文使用夹角余弦相似度计算公式[15]，并结合以上分析，给出相似度计算公式，如式(4)所示：

$$CD(Ct_i,P_i)=\frac{\sum Ct_i*p_j(p_j\geqslant threshold)}{\sqrt{\sum Ct_i(s_j)(s_j\in S_f)}} \tag{4}$$

测试用例 t_i 的覆盖向量为 $Ct_i=(x_1,x_2,\cdots,x_i,\cdots,x_n)$。其中，变量 x_i 的值为 0 或 1，1 表示该测试用例覆盖了对应的语句 s_i，0 则表示未覆盖，$Ct_j(s_j)$ 表示测试用例覆盖了语句 s_i。DSS 排序策略详细算法如图 2 所示。

```
ALGORITHM:Dynamic Similarity Sort
DSS(R,Sf,Ek)
1. 输入:余下测试用例集 R 的上一版本覆盖信息,failed 测试用例集
   Tf 覆盖的语句集 Sf(初始 Sf 为第一个 failed 测试用例覆盖的语句
   集),以及怀疑度公式 Ek
2. 输出:优先执行的测试用例 t
3. for each sj∈Sf
4.   if pj≥threshold
5.     pj←Suspiciousness(sj,tj,Ek)
6.   else
7.     pj← 0
8.   end if
9. end for// 更新 P 向量
10. for each ti∈R
11.   for each pj≥threshold and sj∈Sf
12.     CD(Cti,Pi)←Σk*pj / √k
13.   //计算余下测试用例的覆盖向量与 P 向量的相似度
14.   end for
15. end for
16. Pick t∈R s.t. ∀(t'∈R)(t'≠t)(CD(Ct',P)≤CD(Ct,P))
17. //选择相似度最大的测试用例 t
18. R←R-{t''|t''∈R∧t≠t∧(CD(Ct,P)=CD(Ct'',P)‖CD(Ct'',
    P)==0)}
19. //根据准则一、二排除测试用例
20. Execute t and update tpi,tfi,cpi and cfi
21. if t fail
22.   Sf←St∧Sf//St表示 t 所覆盖的语句集
23.   Tf← t
24. end if
25. return t
```

图 2　动态相似度排序算法

DSS 以余下测试用例集 R 的上一版本覆盖信息、failed 测试用例集 T_f 覆盖的语句集 S_f（初始 S_f 为第一个 failed 测试用例覆盖的语句）以及怀疑度公式 E_k 为输入；每次从 R 中选取一个优先执行的测试用例 t 为输出。当检测到第一个 failed 测试用例时，DSS 按照式(3)计算 P 向量中对应位置 p_j 的值（p_j 初始值为 1，*threshold* 值为 0.1），得到当前 P 向量，如算法中第 3－9 行所示；然后按照式(4)计算 R 中测试用例与 P 向量之间的相似度，如算法中第 10－15 行所示；选择相似度最大的测试用例 t 优先执行，同时根据排除准则，约减相应测试用例，如算法中第 16－19 行所示；然后更新执行通过的测试用例总数 tp_i，执行 failed 测试用例总数 tf_i，以及 $cp_i(s_j)$ 和 $cf_i(s_j)$，其分别表示测试用例覆盖 s_j 且执行结果为 passed 或 failed 的数量；最后如果测试用例 t 的执行结果为 failed，则 t 覆盖的语句集 S_t 与 S_f 取交得到新的 S_f，该轮选择结束，如算法中第 20－25 行所示。

利用 DSS 计算表 1 中的实例，当 t_1 执行结果为 failed 时的语句怀疑度值如表 1 第 6 列所示。t_2、t_3 和 t_4 相似度分别是 0.70、0.82、0.70，则优先执行 t_3，执行结果为 passed。根据此时执行的测试用例计算相应语句的怀疑度，并更新 P 向量的值，如表 1 第 7 列所示。然后计算 t_2 和 t_4 的相似度分别是 0.50 和 0.67，可得 t_4 的相似度大于 t_3，则优先执行 t_4，计算语句的怀疑度值如表 1 所列。可得只有包含错误的语句 s_7 的怀疑度值为 1，且为最高。

4　实验设计与分析

4.1　研究问题

为了验证本文方法的有效性，针对以下 3 个研究问题进行了实验验证。

(1)本文提出的基于动态相似度的错误定位优先排序方法是否可以提高错误定位的准确度？

(2)在保证错误定位准确度的前提下，本文提出的方法对测试用例集约减能力如何？

(3)在检测错误阶段，采用不同测试用例优先排序算法对后续错误定位效果的影响如何？

4.2　实验设计

如图 1 所示，错误定位优先排序主要包括以检测错误为目标的优先排序和以定位错误为目标的优先排序。在第一阶段，实验选用了 3 种常用的测试用例优先排序算法 RND、ADD 和 ART-maximize the minimum 来研究不同算法对后续错误定位准确度的影响。为了证明本文方法能有效提高错误定位的准确度，在第二阶段检测到第一个 failed 的测试用例时，实验分别采用引入 P 向量的方法（即基于动态相似度排序的方法）和未引入 P 向量的排序方法，并根据终止条件停止，然后利用 Tarantula 和 Ochiai 两种错误定位技术定位错误，并对其结果进行比较。实验结果还与随机生成一个测试用例执行序列后执行全部测试用例时定位错误的方法的结果进行对比，并把该方法作为错误定位有效性比较的基准方法。为了评估本文方法的测试用例约减能力，同时比较了定位错误时所执行的测试用例数目。

4.3　实验对象

本文在 6 个来自 SIR 库[1)] 的基准 C 语言程序的多个错误版本基础上，排除了一些多错误版本和没有包含执行 failed 的测试用例以及一些因语句遗漏而引发错误的程序版本。各基准程序、测试用例及其他相关信息如表 2 所列。实验中的覆盖信息是利用 gcov 工具在 Linux 环境下生成的。

表 2　实验程序

程序	错误版本数	使用版本数	LOC	测试用例数
tcas	42	20	173	1608
schedule	6	4	412	2650
replace	32	18	564	5543
totinfo	24	15	565	1052
printtokens2	9	6	570	4055
space	39	16	6199	13585

4.4　度量方法

评价一种错误定位技术有效性的方案有很多，通常以检查到包含错误的语句时检查的语句数目多少来衡量错误定位

1) http://sir.unl.edu/portal/index.php

技术的优劣，即以测试人员按照语句怀疑度大小顺序依次检查代码，直到找到错误语句时需检查的语句数目为衡量标准。本实验使用了类似的度量方法[8]，公式定义如下：

$$W(c_d)=\frac{c_d}{M-1}*100\% \tag{5}$$

其中，$W(c_d)$用来衡量开发人员找到真正错误代码时必须检查的语句数目百分比，即代码检查率。M是程序可执行的语句数，c_d是错误语句的Rank值。因为不同语句可能拥有相同的怀疑度值及Rank值，所以c_d表示一个平均值[8]，定义如下：

$$c_d=\frac{\{k:Sup(s_i)>Sup(s_j)\}+|\{k:Sup(s_i)\geqslant Sup(s_j)\}|-1}{2} \tag{6}$$

其中，$Sup(s_i)$表示语句s_i的怀疑度，同理$Sup(s_j)$是语句s_j的怀疑度；k是一个整数，表示满足式(6)中条件的个数。

为进一步比较本文方法与其他方法的错误定位准确度，从代码检查率入手，定义了代码检查率约减公式：

$$RW=\frac{W(c_d(100\%))-W(c_d(n\%))}{W(c_d(100\%))}*100\% \tag{7}$$

RW表示利用错误定位优先排序方法与执行全部测试用例方法相比，在定位错误时代码检查率的约减百分比。其中$W(c_d(n\%))$表示利用错误定位优先排序方法使用$n\%$的测试用例定位错误时的代码检查率，$W(c_d(100\%))$表示基准方法执行全部测试用例时的代码检查率。代码检查率的约减百分比RW越高，错误定位时代码检查率越低。

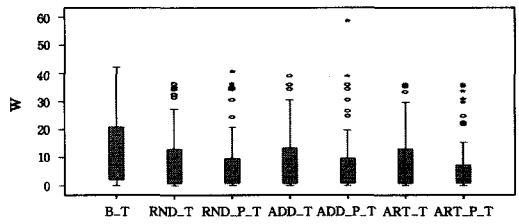
为观察不同错误定位优先排序方法对测试用例的约减能力，本实验评估了将3种测试用例优先排序算法和2种怀疑度公式相结合来定位错误时所使用的测试用例数约减率[15,16]，其约减公式为：

$$Reduction=(1-\frac{size\ of\ reduced\ test\ suite}{size\ of\ unreduced\ test\ suite})*100\% \tag{8}$$

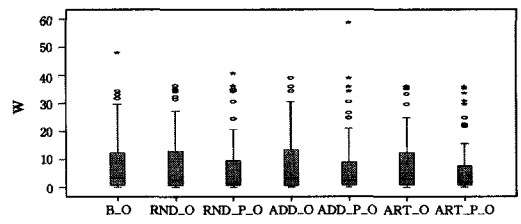
其中，*Reduction*表示测试用例约减率，体现了不同方法测试用例的约减能力；*size of reduced test suite*代表约减后使用的测试用例数；*size of unreduced test suite*表示原有测试用例数。

4.5 结果分析

研究问题1：为了验证本文方法错误定位的准确度，DSS引入P向量的方法分别与未引入P向量的方法以及基准方法错误定位结果进行了对比。实验分别采用RND、ADD和ART算法，检测到错误后，结合相应的排序策略，在本文使用的所有程序版本上进行了实验，结果如图3所示。其中纵坐标表示利用式(5)所得的代码检查率，横坐标表示不同错误定位优先排序方法；名称中字母B表示基准方法，RND、ADD和ART分别表示第一阶段采用的相应检测错误算法，字母P则表示第二阶段采用引入P向量的DSS排序策略，否则采用未引入P向量的排序策略；字母T和O分别表示使用*Tarantula*和*Ochiai*怀疑度公式进行错误定位。如ART_P_T表示采用ART算法检测错误，当发现第一个failed测试用例时，利用DSS引入P向量的排序策略结合*Tarantula*怀疑度公式定位错误的方法。



(a) Tarantula



(b) Ochiai

图3 每种错误定位方法的代码检查率

图3(a)和图3(b)是基于*Tarantula*和*Ochiai*两种怀疑度公式错误定位的结果。从图3(a)可以看出，基准方法B_T代码检查率中位数在8%左右，而未引入P向量的排序策略结合*Tarantula*错误定位方法RND_T、ADD_T和ART_T代码检查率中位数分别在3%、4%和3%左右，引入P向量的方法RND_P_T、ADD_P_T和ART_P_T分别在2%、3%和2%左右。未引入P向量的和引入P向量的方法代码检查率中位数都在3%左右，都比较低，能较大程度地约减错误定位时检查代码的数量，而本文提出的引入P向量的方法绝大多数情况下错误定位的代码检查率都优于其他方法。从图3(b)可以看出，结合怀疑度公式*Ochiai*，未引入P向量的方法RND_O、ADD_O和ART_O代码检查率中位数都在3%左右，与基准方法B_O的4%比较，优势并不明显，但引入P向量的方法RND_P_O、ADD_P_O和ART_P_O代码检查率中位数都在2%以内，与基准方法比较同样占有明显优势。

为了进一步观察各种方法代码检查率的差别，本文利用式(7)计算了未引入P向量和引入P向量的相应方法与基准方法错误定位相比的代码检查率约减百分比，如表3所列。其中，第一、二列分别是未引入P向量的相应方法和代码检查率约减百分比；随后两列是引入P向量的方法和代码检查率约减百分比，负数表示比基准方法多检查的百分比。

表3 代码检查率的约减百分比*RW*

未引入P向量方法	代码检查率的约减百分比RW	引入P向量方法	代码检查率的约减百分比RW
RND_O	−1.78%	RND_P_O	14.97%
ADD_O	−9.35%	ADD_P_O	8.70%
ART_O	1.01%	ART_P_O	16.80%
RND_T	27.29%	RND_P_T	34.01%
ADD_T	23.45%	ADD_P_T	30.29%
ART_T	26.90%	ART_P_T	36.03%
Average	11.25%		23.47%

从表3可以看出，DSS引入P向量的方法占有明显优势，最好情况能提高36.03%，最少能达到8.07%。未引入P向量的方法最好情况能提高27.29%，而结合*Ochiai*怀疑度计算公式，未引入P向量的方法改进并不明显，在某些情况下甚至低于基准的方法，如RND_O和ADD_O分别比基准的方法多检查1.78%和9.35%的代码。这是因为*Ochiai*怀疑度公式没有考虑未覆盖相应语句执行passed的测试用例对错误定位的影响，另外，现存的方法没有考虑具体语句的区

别。从表 3 最后一行可得出，未引入 P 向量的方法代码检查率约减百分比平均提高了 11.25%，引入 P 向量的方法平均提高了 23.47%。

综上所述，DSS 引入 P 向量的方法与相应未引入 P 向量的方法相比，代码检查率约减百分比平均提高了 12.22%，能进一步提高错误定位的准确度，改善错误定位效果，所以本文提出的引入 P 向量的相似度排序策略更有利，能较多地约减错误定位时检查代码的数目。

研究问题 2：为验证测试用例约减能力，实验对相应方法错误定位时执行的测试用例数目进行了对比，如图 4 所示。其中，纵坐标表示执行测试用例的百分比，横坐标表示相应方法。

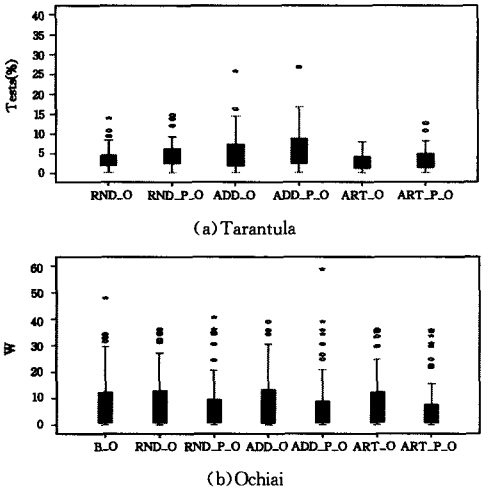


图 4　每种错误定位方法执行测试用例的百分比

从图 4(a)中可以看出，未引入 P 向量的方法 RND_T、ADD_T 和 ART_T 在执行测试用例百分比中位数分别为 3%、4%、3%左右，引入 P 向量的方法 RND_P_T、ADD_P_T 和 ART_P_T 分别为 4%、5%、3%左右；从图 4(b)可以得到相同结论，引入 P 向量和未引入 P 的方法平均执行测试用例数都在 5%左右，能较多地约减执行的测试用例数目。DSS 引入 P 向量的方法比相应未引入 P 向量的方法执行测试用例数目略微偏多。

为了进一步比较各种方法执行测试用例数目的详细情况，实验利用式(8)计算了测试用例约减率，详细数值如表 4 所列。其中第一、二列分别表示未引入 P 向量的相应方法和测试用例约减率，第三、四列是引入 P 向量的相应方法和约减率。从表 4 可得，未引入 P 向量的方法测试用例约减率最好能提高 97.13%，最差能达到 94.89%，平均测试用例约减率为 96.09%；引入 P 向量的方法最好能提高 96.60%，最差能达到 93.72%，平均测试用例约减率为 95.23%(如表 4 最后一行所示)，引入 P 向量的方法比未引入 P 向量的方法平均多执行 0.86%的测试用例。

表 4　平均测试用例约减率

未引入 P 向量方法	测试用例约减率	引入 P 向量方法	测试用例约减率
RND_O	96.24%	RND_P_O	95.35%
ADD_O	94.89%	ADD_P_O	93.85%
ART_O	97.13%	ART_P_O	96.60%
RND_T	96.24%	RND_P_T	95.30%
ADD_T	94.89%	ADD_P_T	93.72%
ART_T	97.13%	ART_P_T	96.57%
Average	96.09%		95.23%

综上所述，基于 DSS 排序策略的方法对测试用例的约减能力稍逊于未引入 P 向量的方法，但测试用例约减率平均能达到 95%以上，也能较多地约减执行的测试用例数目。此外，引入 P 向量的方法比未引入 P 向量的方法平均多执行 0.86%的测试用例，而错误定位代码检查率约减百分比平均提高了 12.22%，在错误定位准确度上占有明显优势。

研究问题 3：为观察不同测试用例优先排序算法对后续定位错误效果的影响，在检测错误阶段，本实验采用了 3 种测试用例优先排序算法，即 RND、ADD 和 ART。从图 3 可得，利用这 3 种测试用列优先排序算法检测到的第一个 failed 测试用例，然后结合相应针对错误定位的排序策略都能有效降低错误定位的代码检查率；基于不同测试用例优先排序算法检测到错误，然后定位错误，其定位错误的效果也有所不同，如表 3 所列。其中结合 Ochiai 怀疑度公式，未引入 P 向量的方法中，采用 ART 检测错误，对后续错误定位更有利，能提高1.01%的代码检查率；结合 Tarantula 公式时，采用 RND 对后续定位错误更有利，能达到 27.29%的代码检查率约减百分比。基于 DSS 引入 P 向量的方法中，都是 ART 算法表现最为优越，分别能达到 16.80%和 36.03%的代码检查率约减百分比。

基于这 3 种测试用例优先排序算法进行错误定位时，其测试用例约减能力的分析如表 4 所列。3 种算法中，测试用例约减能力最强的是基于 ART 的方法，其次是 RND，最后是基于 ADD 的方法。

综上所述，基于不同的测试用例优先排序算法在错误定位和测试用例约减能力上都表现出较好的效果，其中基于 ART 的方法无论是在错误定位代码检查率约减还是在测试用例约减率方面都表现最优。

结束语　错误定位优先排序方法通过优化测试用例的执行顺序，来尽可能地提高错误定位的准确度。本文对错误定位优先排序方法中的相似度计算进行了改进，提出了一种基于动态相似度的错误定位优先排序方法。在相似度计算时，引入了语句变化的怀疑度值，利用语句怀疑度与执行测试用例之间的关系，选择对错误定位更有利的测试用例优先执行，提高了相似度计算的有效性。实验表明，本文的方法能提高错误定位准确度，约减测试用例数目。实验同时表明在检测错误阶段，利用不同的测试用例优先排序算法检测错误对后续定位错误有一定的影响。其中使用 ART 的方法不管是在错误定位准确度还是测试用例约减能力上都表现最优。

本文方法能有效改善定位错误效果。在未来的工作中，将把该方法运用到多错误程序中，并评估该方法的性能，以及分析 *threshold* 的取值对错误定位准确度的影响。

参 考 文 献

[1] Yoo S, Harman M, Clark D. Fault Localization Prioritization: Comparing information-theoretic and coverage-based approaches [J]. ACM Transactions on Software Engineering and Methodology(TOSEM), 2013, 22(3): 19

[2] Chen Xiang, Chen Ji-hong, Ju Xiao-lin, et al. Survey of Test Case Prioritization Techniques for Regression Testing[J]. Journal of Software, 2013, 24(8): 1695-1712(in Chinese)
陈翔，陈继红，鞠小林，等. 回归测试中的测试用例优先排序技术述评[J]. 软件学报，2013，24(8)：1695-1712

[3] Rothermel G, Untch R H, Chu C. Test case Prioritization: An empirical study[C]// IEEE International Conference on Software Maintenance, 1999(ICSM'99). IEEE, 1999: 179-188

[4] Cao He-ling, Jiang Shu-juan, Ju Xiao-lin. Survey of Software Fault Localization[J]. Computers Science, 2014, 41(2): 1-6(in Chinese)
曹鹤玲,姜淑娟,鞠小林. 软件错误定位研究综述[J]. 计算机科学,2014,41(2):1-6

[5] Wong W E, Debroy V. A survey of software fault localization: UTDCS-45-09[R]. Department of Computer Science, University of Texas at Dallas, 2009

[6] Jiang B, Zhang Z, Tse T H. How well do test case prioritization techniques support statistical fault localization[C]// 33rd Annual IEEE International Computer Software and Applications Conference, 2009(COMPSAC'09). IEEE, 2009, 1: 99-106

[7] Jiang B, Chan W K. On the integration of test adequacy, test case prioritization, and statistical fault localization[C]// 2010 10th International Conference on Quality Software (QSIC). IEEE, 2010: 377-384

[8] Gonzalez-Sanchez A, Piel é, Abreu R. prioritizing tests for software fault diagnosis[J]. Software: Practice and Experience, 2011, 41(10): 1105-1129

[9] Gonzalez-Sanchez A, Abreu R, Gross H G. Prioritizing tests for fault localization through ambiguity group reduction[C]// 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2011: 83-92

[10] Xie X, Chen T Y, Kuo F C. A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization[J]. ACM Transactions on Software Engineering and Methodology (TOSEM), 2013, 22(4): 31

[11] Rothermel G, Untch R H, Chu C. Prioritizing test cases for regression testing[J]. IEEE Transactions on Software Engineering, 2001, 27(10): 929-948

[12] Yoo S, Harman M. Regression testing minimization, selection and prioritization: a survey[J]. Software Testing, Verification and Reliability, 2012, 22(2): 67-120

[13] Jiang B, Zhang Z, Chan W K. Adaptive random test case prioritization[C]// 24th IEEE/ACM International Conference on Automated Software Engineering, 2009(ASE'09). IEEE, 2009: 233-244

[14] Xue X, Namin A S. How Significant is the Effect of Fault Interactions on Coverage-Based Fault Localizations? [C]// 2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. IEEE, 2013: 113-122

[15] Wang Ke-chao, Wang Tian-tian, Su Xiao-hong, et al. Test Case Selection for Improving the Effectiveness of Software Fault Localization[J]. Journal of Computer Research and Development, 2014, 51(4): 865-873(in Chinese)
王克朝,王甜甜,苏小红,等. 面向有效错误定位的测试用例优选方法[J]. 计算机研究与发展,2014,51(4):865-873

[16] Yu, Yan-bing, Jones J A, et al. An empirical study of the effects of test-suite reduction on fault localization[C]// Proceedings of the 30th International Conference on Software Engineering. ACM, 2008: 201-210

(上接第 192 页)

[4] Johnson M, Rosebrugh R. Fibrations and universal view updatability[J]. Theoretical Computer Science, 2007, 388: 109-129

[5] Johnson M, Rosebrugh R, Wood R J. Lenses, fibrations and universal translations[J]. Mathematics Structure in Computer Science, 2012, 22: 25-42

[6] Tews H. Coalgebra method for object-oriented specification[D]. Dresden, Germany: Institute Theoretische Informatik, Technischen Universiy Dresden, 2002

[7] Ghani N, Johann P, Fumex C. Generic fibrational induction[J]. Logical Methods in Computer Science, 2012, 8(2): 1-27

[8] Miao De-cheng, Xi Jian-qing, Jia Lian-yin, et al. Formal language algebraic model [J]. Journal of South China University of Technology (Natural Science Edition), 2011, 39(10): 74-78(in Chinese)
苗德成,奚建清,贾连印,等. 一种形式语言代数模型[J]. 华南理工大学学报(自然科学版),2011,39(10):74-78

[9] Hermida C, Jacobs B. Structural induction and coinduction in a fibrational setting[J]. Information and Computation, 1998, 145(2): 107-152

[10] Barr M, Wells C. Category theory for computing science[M]. NewYork: Prentice-Hall, 1990: 252-270

[11] He Wei. Category theory[M]. Beijing: Science Press, 2006: 23-69(in Chinese)
贺伟. 范畴论[M]. 北京:科学出版社,2006:23-69

[12] Ghani N, Johann P, Fumex C. Indexed induction and coinduction, fibrationally[J]. Logical Methods in Computer Science, 2013, 9(3-6): 1-31

[13] Hermida C. Fibrations, Logical Predicates and Related Topics [D]. Edinburgh, UK: University of Edinburgh, 1993

[14] Su Jin-dian, Yu Shan-shan. Coinductive data types and their applications in programming languages[J]. Computer Science, 2011, 38(11): 114-118(in Chinese)
苏锦钿,余珊珊. 程序语言中的共归纳数据类型及其应用[J]. 计算机科学,2011,38(11):114-118

[15] Hagino T. A categorical programming language [D]. Edinburgh, UK: Laboratory for Foundations of Computer Science, Dept of Computer Science, University of Edinburgh, 1987

[16] Nogueira P, Moreno-Navarro J. Bialgebra views: a way for polytypic programming to cohabit with data abstract[C]// Proceedings of the ACM SIGPLAN Workshop on Generic Programming. ACM New York, NY: 2008: 61-73

[17] Poll E. Subtyping and inheritance for categorical datatypes[J]. RIMS Lecture Notes, 1998, 1023: 112-125

[18] Hinze R. Reasoning about codata [J]. Lecture Notes in Computer Science, 2010, 6299: 42-93

[19] Gimenez E, Casteran P. A Turorial on Co-inductive Types in Coq[OL]. May 1998. http: www. labri. fr/perso/casteran/RecTutorial. pdf

[20] Vene V. Categorical programming with inductive and coinductive types[D]. Tartu, Estonia: University of Tartu, 2000