基于定值-引用链的测试用例优先级排序算法

潘丽丽[1] 王天锷[2] 秦姣华[1] 向旭宇[1]
(中南林业科技大学计算机与信息工程学院 长沙 410004)[1]
(湖南省送变电建设公司调试所 长沙 410017)[2]

摘 要 测试用例优先级排序作为一种高效实用的回归测试技术,通常以测试用例的覆盖度作为优先级排序的量化指标,忽略了测试用例的其他测试性能。针对该问题,提出一种基于DU链的测试用例优先级排序算法。该算法综合考虑测试用例的DU链覆盖度和回归测试的错误检测能力,对测试用例优先级进行量化。与已有算法相比,该算法基于数据流覆盖,充分利用了测试执行的历史信息和程序模块的耦合信息,在排序过程中动态计算测试用例的优先级量化值。实验结果表明,采用优先级排序算法的测试用例集能在测试过程中以较短的时间发现更多的错误,有效地提高了回归测试的检错效率。

关键词 回归测试,测试用例,优先级排序,定值-引用链,错误检测率
中图法分类号 TP311.5 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.04.038

Test Case Prioritization Based on DU Chains

PAN Li-li[1] WANG Tian-e[2] QIN Jiao-hua[1] XIANG Xu-yu[1]
(College of Computer Science and Information Technology, Central South University of Forestry and Technology, Changsha 410004, China)[1]
(The Commission Institute, Hunan Electric Power Transmission and Substation Construction Company, Changsha 410017, China)[2]

Abstract Test case prioritization is an effective and practical technique of regression testing. Yet this technique is quite limited in a way that it prioritizes testing cases based on test-requirement coverage only and ignores many other testing factors. To improve the performance, this paper presented a new test case prioritization algorithm based on DU chain. The algorithm combines the DU-chain coverage and fault detection rate as the test-case quantitative factors. Compared with existing algorithms, the new algorithm makes use of information from executed testing and modules coupling, and dynamically calculates priority quantitative value for every test case. The experimental resucts show that the new prioritization algorithm is helpful to detect more faults in a shorter time.

Keywords Regression testing, Test case, Prioritization, DU chain, Rate of fault detection

1 引言

目前,测试用例优先级排序技术主要是基于测试用例的覆盖率。人们最初通过计算测试用例的覆盖率进行优先级排序[1]。随后,研究者们或从语句覆盖、分支覆盖和数据流覆盖等需求的覆盖角度,或从静态分析角度,或从特定的应用角度提出了多种测试用例优先级排序方法[2-6],实验研究结果表明这些优先级排序算法都能有效提高回归测试的检错效率。

已有的优先级排序方法对测试用例的需求覆盖技术进行了有效的分析和利用,但没有基于DU链路径分析回归测试的历史信息以及测试用例非覆盖度的其他特征信息。为此,本文基于定值-引用链覆盖的测试用例集,采用测试用例的覆盖度和回归测试检错能力作为优先级量化因子,给出测试用例优先级量化公式,在此基础上给出了静态优先级排序算法和动态优先级排序算法。实验数据表明,动态优先级排序算法的检错效率更高。

2 基本概念

2.1 基于DU链的覆盖分析

DU链路径覆盖是一种常用的数据流测试覆盖标准。为了描述程序中的DU链,给出以下相关定义。

定义1 节点表示程序的一条执行指令,用 n 表示节点,N 表示节点集。

声明语句、赋值语句、判断语句等都可以看作一个节点 n_i

到稿日期:2015-11-30 返修日期:2016-02-25 本文受国家自然科学基金项目(61304208),湖南省自然科学基金重点项目(13JJ2031),湖南省自然科学基金项目(13JJ4087),湖南省教育厅科学研究项目(16C1659),中南林业科技大学教学改革研究项目(1020208),湖南省科技项目(2014SK2025),湖南省情与决策咨询研究课题(2013BZZ54),湖南省教育科学"十二五"规划项目(XJK013CXX014)资助。
潘丽丽(1977-),女,博士,副教授,主要研究方向为软件测试、图像检索,E-mail:lily_pan@163.com;王天锷(1975-),男,硕士,高级工程师,主要研究方向为智能工程;秦姣华(1973-),女,博士,教授,主要研究方向为信息安全;向旭宇(1972-),男,博士,教授,主要研究方向为网络安全。

$(1\leqslant i\leqslant|N|)$，$|N|$表示节点集中的节点个数。

定义 2 变量 v 的定义节点集：$def(v)=\{n|$ the node n that defines the variable $v\}$；

变量 v 的引用集：$use(v)=\{n|$ the node n that uses the variable $v\}$。

定义 2 中变量的定义集不仅包括对变量的赋值，变量的类型说明及函数入口对参数的说明都可看作是对变量的定义。

定义 3(DU 链) 给定节点 a 和 b，如果 a 定义了变量 v，之后节点 b 引用 v，即 $a\in def(v)$，$b\in use(v)$，则称 a 和 b 间构成变量 v 的 DU 链，表示为 aDUb，即一个三元组$\langle a,b,v\rangle$。

DU 链路径覆盖是一种常用的白盒测试方法，它通过覆盖变量的定义引用路径以达到对程序进行测试的目的。该方法能够有效地发现程序中隐藏的错误，但是 DU 链路径相当多，特别是在大型程序中实现完整的 DU 链路径测试几乎是不可能的；而基于方法的 DU 链路径覆盖分析技术可以大大缩减 DU 链路径规模，使得 DU 链路径覆盖技术现实可行。

定义 4(方法包含 DU 链路径) 对于给定的 DU 链 du，给定方法 M，如果该 DU 链出现在方法 M 中，则 M 包含 du，表示为 $du\in DU(M)$。

2.2 测试用例优先级排序技术

测试用例优先级排序技术根据约定的方法计算测试用例对测试目标的贡献程度，根据贡献程度的大小对测试用例的执行顺序进行优先级排序，以尽快实现测试的目标。

测试用例优先级技术的核心问题是如何确定和表述测试用例的重要程度[2]。目前，主要是依据测试用例的覆盖率进行测试用例优先级排序，即优先运行覆盖能力强的测试用例。显然，传统的测试用例优先级技术仅仅考虑了测试用例的覆盖能力，忽略了影响测试用例的错误检测能力的其他因素[7]。

3 测试用例优先级排序算法

3.1 测试用例优先级量化

本文将测试用例的覆盖率和测试用例的回归测试检错能力作为测试用例优先级量化的两个重要影响因子。为了便于描述，给出以下相关定义。

定义 5(测试覆盖矩阵) 测试用例集 $T=\{t_1,t_2,\cdots,t_n\}$，变更的 DU 链集即测试需求集 $DU=\{du_1,du_2,\cdots,du_m\}$，则测试覆盖矩阵 tCM 是一个 $m\times n$ 阶矩阵，当 t_i 在执行过程中执行到了 DU 链 du_j 时，元素 $\delta_{ij}=1$，否则 $\delta_{ij}=0$。

定义 6(测试用例覆盖的 DU 链集) 测试用例 t_i 覆盖的 DU 链集 $DU(t_i)$($DU(t_i)\in DU$)是一个 DU 链覆盖集，其中包含执行测试用例 t_i 时所覆盖的 DU 链称为测试用例覆盖的 DU 链集。

定义 7(DU 链的测试用例集) du_j 的测试用例集 $T(du_j)$($T(du_j)\in T$)是一个测试用例集，为测试用例集 T 中所有能够覆盖 DU 链 du_j 的所有测试用例称为 DU 链的测试用例集。

3.1.1 测试用例的 DU 链覆盖率

测试用例 t_i 覆盖 DU 链的能力是指测试用例 t_i 覆盖 DU 链集中的 DU 链数量$|DU(t_i)|$。将测试用例 DU 链覆盖能力应用于测试用例优先级排序中，其 DU 链覆盖率的计算公式为：

$$F_i^{DN}=\frac{N_i}{\max\{N\}} \tag{1}$$

其中，N_i 为测试用例 t_i 覆盖的 DU 链的数量$|DU(t_i)|$；$\max\{N\}$为最大的 N_i。式(1)将测试用例对 DU 链的覆盖率量化到 0～1 的数值区间。

3.1.2 测试用例的回归测试检错能力

测试经验表明，若一个模块出现错误，很可能会引起与该模块关联的其他模块出错，与其他模块关联度高的模块会使得错误迅速扩散。程序的耦合性是对程序结构中各个模块之间相互关联程度的一种度量，它取决于各个模块之间接口的复杂程度，本文采用模块的扇入系数来描述模块间的耦合性。扇入系数指直接调用该模块的上级模块的个数。面向对象程序的单元测试是基于类的测试，类中的调用模块通常是指方法[5]。

(1)基于方法的 DU 链检错能力

若已知 DU 链 du_j，其包含在方法 M_k 中，则该 DU 链基于方法的检错能力量化为：

$$F_j^M=\frac{ME_k}{\max\{ME\}}\times\frac{CM_k}{\max\{CM\}} \tag{2}$$

其中，F_j^M 为 DU 链 du_j 的检错指标，且在回归测试中 M_k 方法至少检测到一个错误；ME_k 为回归测试时 M_k 检查的错误个数；$\max\{ME\}$指在回归测试时所有被测试方法检测错误个数的最大值；CM_k 为方法 M_k 的扇入值；$\max\{CM\}$是指在回归测试中至少检测出了一个错误的方法扇入值中的最大扇入值。式(2)将基于方法的 DU 链的检错指标量化到 0～1 的取值范围。

(2)基于 DU 链覆盖的测试用例检错能力

测试用例 t_i 的 DU 链检错指标是指测试用例 t_i 覆盖的 DU 链集的检错能力评估，如式(3)所示：

$$F_i^T=\frac{\sum_{du_j\in du(t_i)}F_j^M}{n} \tag{3}$$

其中，n 为 t_i 覆盖的 DU 链的个数。式(3)将测试用例的 DU 链覆盖检错能力值约束在 0～1 范围内。

3.1.3 测试用例优先级量化评估

本文综合测试用例的 DU 链覆盖率和回归测试检错能力两个指标对测试用例的优先级进行量化。

$$F_i=\varphi_1\times F_i^{DN}+\varphi_2\times F_i^T \tag{4}$$

其中，F_i^{DN} 表示测试用例 t_i 的 DU 链覆盖率；F_i^T 表示测试用例 t_i 的回归测试检错能力；φ_1 和 φ_2 分别为 F_i^{DN} 和 F_i^T 对应的权重系数，且 $\varphi_1+\varphi_2=1$，其取值可以根据测试的实际情况进行调整。

3.2 测试用例优先级排序算法

测试用例的优先级度量主要由测试用例的 DU 链覆盖率和其回归测试检错能力决定。根据 3.1 节中给出的定义和优先级量化公式对测试用例进行优先级量化，并依据量化值赋予每个测试用例执行顺序。

由于每一个测试用例在排序的过程中对未覆盖的 DU 链而言其贡献度是不一样的，因此在选择测试用例时考虑已经

排序的测试用例，动态调整并重新计算未排序测试用例的优先级量度，即：根据已排序的测试用例，将 DU 链集划分为已覆盖的 DU 链集和未覆盖的 DU 链集，未排序的测试用例根据其未覆盖的 DU 链重新计算优先级量化值，选择优先级量化值最高的测试用例，如此反复，直到所有的 DU 链都被覆盖为止。该方法既可以保证 DU 链覆盖的完整性，动态调整测试用例的优先级量化计算，同时也使优先级量化值较高的测试用例获得优先的测试执行顺序。本文将这种方法称为动态优先级排序算法，具体的算法描述如算法 1 所示。

算法 1　动态优先级排序算法

```
输入：回归测试用例集 T；变更的 DU 链集 DU；测试用例 DU 链；覆盖
      矩阵 tDUCM
输出：优先级排序后的测试用例集 T'
变量：testDU_UC 为待覆盖的 DU 链；test_UP 为待排序的测试用例
Priority(T)； // 根据式(4)对测试用例进行优先级量化
testDU_UC=DU；
test_UP=T；
while(test_UP){ //动态优先级排序
  t1=max(test_UP)；test_UP=test_UP−t1；
  Equeue(t1)；
  if(testDU_UC){
    DU_NC=DU(t1)∩testDU_UC
    if(DU_NC){
      //遍历测试用例 t1 新覆盖的 DU 链
      for each DU chain du_i in DU_NC{
        for each test case t_k in T(du_i){
      //剔除 DU 链 du_i，重新计算测试用例 t_k 的优先级排序值
          ModifyPriority(t_k)；
        }
      }
      testDU_UC=testDU_UC-DU_NC；
    }
  }
  else{
    for the number of test_UP{
      t1=max(test_UP)；test_UP=test_UP−t1；
      Equeue(t1)；
    }
  }
}
```

4　实验与分析

4.1　优先级排序方法度量标准

测试用例优先级排序作为一种重要的回归测试技术，其目的是尽快实现测试目标，尽早发现程序中的错误，提高错误检测率。APFD(Average Percentage Faults Detected)是较早使用且较为普遍的一种优先级排序度量标准[2]。APFD 称为缺陷检测加权平均百分比，即若测试用例集 T 包含 n 个测试用例，该测试用例集能够检测出的错误集 F 中包含 m 个错误，则测试用例集 T 的一个顺序集 T' 的 APFD 值定义为：

$$APFD=1-\frac{TF_1+TF_2+\cdots+TF_m}{n\times m}+\frac{1}{2n} \tag{5}$$

其中，TF_1 为顺序集 T' 中第一个检测出错误 i 的测试用例在 T 中的位置。APFD 的取值范围为 0～1。

由式(5)可知，APFD 值越大表示测试用例集的执行顺序错误检测率越高。

4.2　实验结果

为了检测测试用例优先级排序算法的有效性，本文选取了 5 个不同功能和规模且自主开发的 Java 程序作为被测程序进行实验。采用 JUnit 编写测试用例，且完整覆盖基于方法的 DU 链集，同时在被测程序中人工植入互不干扰且数量不等的错误，回归测试信息由多次编程调试测试记录的数据构成。针对这 5 个被测试程序，分别执行随机测试和动态优先级排序算法进行两组测试实验，具体为：

(1)执行未排序的测试用例集 T；

(2)对原始测试用例集采用动态优先级排序算法进行优先级排序，得到排序的测试用例集 T'。

每一个被测试程序的规模、方法数、DU 链数、测试用例数、植入的错误数以及两组不同测试用例顺序集执行后得到的 APFD 值如表 1 所列。

表 1　不同测试集执行后的 APFD 数据表

Program \ Item	Lines	Methods	DU chains	Test cases	Faults	APFD of T	APFD of T'
P_1	18	4	8	9	2	0.38	0.95
P_2	25	1	7	3	2	0.33	0.67
P_3	48	5	10	11	5	0.36	0.74
P_4	77	1	36	21	11	0.42	0.71
P_5	132	13	73	41	15	0.46	0.69

测试结果表明，程序 P_1-P_5 的测试用例集经过动态优先级算法排序后其 APFD 值均是最高的，明显高于未排序的测试用例集的 APFD 值。这说明排序后的测试用例集在错误检测效率方面明显优于未排序的测试用例集，可以更早、更快地发现错误。

结束语　本文分析了现有测试用例集排序算法的特点，基于 DU 链覆盖和回归测试信息，提出了一种新的测试用例集优先级排序算法。该算法将测试用例的 DU 链覆盖度和回归测试错误检测能力作为测试用例优先级的量化因素，在排序的过程中，根据 DU 链覆盖动态计算测试用例优先级量化值。实验结果表明，采用动态优先级排序算法的测试用例集可以在较短的时间内发现更多的错误，有效提高了错误的检测效率。

参 考 文 献

[1] WONG W E, HORGAN J R, LONDON S, et al. A study of effective regression testing in practice[C]// Proceedings of the 8th IEEE International Symposium on Software Reliablity Engineering, 1977. Albuquerque, New Mexio: IEEE Comp Soc, 1997: 264-274.

[2] ROTHERMEL G, UNTCH R H, CHU C Y, et al. Prioritizing test cases for regression testing[J]. IEEE Transactions on Software Engineering, 2001, 27(10): 929-948.

[3] RUMMEL M J, KAPFHAMMER G M, THALL A. Towards the prioritization of regression test suites with data flow information[C]// Proceedings of the ACM Symposium on Applied

Computing, 2005. Santa Fe, 2005: 1499-1504.

[4] ARAFEEN M J, DO H. Test Case Prioritization Using Requirements-Based Clustering[C]// 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation, 2013. Luembourg: IEEE Conference Publications, 2013: 312-321.

[5] HONG M, DAN H, Z L Mm, et al. A static approach to prioritizing JUnit test case[J]. IEEE Transactions on Software Engineering, 2012, 38(6): 1258-1275.

[6] KE Z, BO J, W K C. Prioritizing Test Cases for Regression Testing of Location-Based Services: Metrics, Techniques, and Case Study[J]. IEEE Transactions on Software Engineering, 2014, 7(1): 54-67.

[7] ALESSANDRO M, MAHFUZUL I, WASEEM A, et al. A Multi-Objective Technique to Prioritize Test Cases[J]. IEEE Transactions on Software Engineering, 2015, 99: 1-22.

(上接第 147 页)

提出的轻量化检测策略在保证检测到相同大型概念的情况下，无论在概念数还是构造时间的性能方面均有大幅度提升，能够有效缩减概念数，降低时间开销。

表 4 轻量化策略生成时间与效率提升

被测程序	完整概念格生成时间/s	轻量化策略生成时间/s	效率提升/%
acct-6.3.2	45.9642	18.2294	60.34
EPWIC-1	3263.1785	2244.0879	31.23
barcode	3759.1561	2054.5517	45.35
copia	1837.3601	848.1254	53.84
findutils	4216.6482	2856.3575	32.26
replace	2487.2165	1287.1022	48.25
userv-client	4332.8168	3060.0865	29.37
which	1897.4587	1305.6717	31.19
bc	5674.0319	1975.6185	65.18
dc	4218.1547	1309.8762	68.95
ed	3824.9426	1409.6520	63.15
indent	5246.4660	3009.2785	42.64
平均	3400.2829	1781.5531	47.65

结束语 针对 MSG 在依赖簇检测准确度方面的不足，本文提出基于 FCA 的依赖簇检测新方法。首先，探讨了如何将源代码中获取的依赖关系抽象和整理成形式背景，搭建了形式概念分析和依赖簇检测的桥梁。从理论层面探讨了概念格构造算法生成所有概念后如何从中选取大型概念，分析和给出了利用大型概念检测依赖簇的方法，并设计了基于形式概念分析的依赖簇检测方法的流程。其次，为提高效率，更有针对性地选取大型概念，提出了基于对象包含度的 FCBO 算法剪枝改进策略，并从理论层面证明了其有效性，同时设计了轻量化策略的流程。最后，为验证新方法及其轻量化检测策略的有效性，选取了 12 个不同规模、不同领域的被测程序进行实验研究。结果表明，本文提出的方法有效地提高了依赖簇检测的准确度，轻量化检测策略不但可以选取与完整概念格相同的大型概念，还能大幅度约减概念数，缩小大型概念的搜索域，降低构造概念格的时间开销。

参 考 文 献

[1] BINKLEY D, HARMAN M. Locating dependence clusters and dependence pollution[C]// Proceedings of the 21st IEEE International Conference on Software Maintenance, 2005 (ICSM'05). IEEE, 2005: 177-186.

[2] ACHARYA M, ROBINSON B. Practical change impact analysis based on static program slicing for industrial software systems [C]// Proceedings of the 33rd International Conference on Software Engineering. ACM, 2011: 746-755.

[3] BESZÉDES A, GERGELY T, JÁSZ J, et al. Computation of tatic execute after relation with applications to software maintenance[C]// IEEE International Conference on Software Maintenance, 2007 (ICSM 2007). IEEE, 2007: 295-304.

[4] SZEGEDI A, GERGELY T, BESZEDES A, et al. Verifying the concept of union slices on Java programs[C]// 11th European Conference on Software Maintenance and Reengineering, 2007 (CSMR'07). IEEE, 2007: 233-242.

[5] HAJNAL A, FORGÁCS I. A demand-driven approach to slicing legacy COBOL systems[J]. Journal of Software: Evolution and Process, 2012, 24(1): 67-82.

[6] BINKLEY D, GOLD N, HARMAN M, et al. Dependence anti patterns[C]// 23rd IEEE/ACM International Conference on Automated Software Engineering. IEEE, 2008: 25-34.

[7] ACHARYA M, ROBINSON B. Practical change impact analysis based on static program slicing for industrial software systems [C]// Proceedings of the 33rd International Conference on Software Engineering. ACM, 2011: 746-755.

[8] BINKLEY D, HARMAN M. Identifying' Linchpin Vertices' That Cause Large Dependence Clusters[C]// Ninth IEEE International Working Conference on Source Code Analysis and Manipulation, 2009 (SCAM'09). IEEE, 2009: 89-98.

[9] BINKLEY D, HARMAN M, HASSOUN Y, et al. Assessing the impact of global variables on program dependence and dependence clusters[J]. Journal of Systems and Software, 2010, 83 (1): 96-107.

[10] HARMAN M, BINKLEY D, GALLAGHER K, et al. Dependence clusters in source code[J]. ACM Transactions on Programming Languages and Systems (TOPLAS), 2009, 32(1): 1-33.

[11] BLACK S, COUNSELL S, HALL T, et al. Fault analysis in OSS based on program slicing metrics[C]// 35th Euromicro Conference on Software Engineering and Advanced Applications, 2009 (SEAA'09). IEEE, 2009: 3-10.

[12] BLACK S, COUNSELL S, HALL T, et al. Using program slicing to identify faults in software[M]// Dagstuhl, Seminar N°. 2005.

[13] ISIAM S S, KRINKE J, BINKLEY D. Dependence cluster visualization[C]// Proceedings of the 5th International Symposium on Software Visualization. ACM, 2010: 93-102.

[14] WILLE R. Restructuring Lattice Theory: An Approach Based on Hierarchies of Concepts[M]// Ordered Sets. Springer Netherlands, 1982: 445-470.