计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2020,29(9):40−46 [doi: 10.15888/j.cnki.csa.007496]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

Evosuite 和 Randoop 单元测试用例生成工具覆盖率对比分析①

杨正卉, 洪　玫, 郭　丹, 王　潇, 刘　芳, 黄小丹

(四川大学 计算机学院 (软件学院), 成都 610065)
通讯作者: 洪　玫, E-mail: hongmei@scu.edu.cn

摘　要: 在软件测试中, 测试用例对被测软件的覆盖率, 是发现软件缺陷的重要前提之一. 本文采用软件工程实验方法, 基于 Defects4J 数据集, 对 Evosuite 和 Randoop 工具在不同的生成时间限制下, 生成的测试用例对程序模块的覆盖率, 程序分支的覆盖率等进行实验分析, 发现当生成时间超过 20 s 时, 虽然 Randoop 生成的测试用例多于 Evosuite, 但 Evosuite 测试用例的覆盖率明显优于 Randoop. 本文同时对影响覆盖率的因素进行了分析. 该研究对于如何使用这两种工具生成高覆盖率的测试用例, 以及对工具的改进具有参考价值.

关键词: 自动化单元测试; 测试用例; 测试覆盖率; Evosuite; Randoop

引用格式: 杨正卉,洪玫,郭丹,王潇,刘芳,黄小丹.Evosuite 和 Randoop 单元测试用例生成工具覆盖率对比分析.计算机系统应用,2020,29(9):40–46. http://www.c-s-a.org.cn/1003-3254/7496.html

Coverage Comparison Analysis of Unit Test Case Generation Tools: Evosuite and Randoop

YANG Zheng-Hui, HONG Mei, GUO Dan, WANG Xiao, LIU Fang, HUANG Xiao-Dan

(College of Computer Science (College of Software Engineering), Sichuan University, Chengdu 610065, China)

Abstract: In the software testing, coverage of test cases is one of the important prerequisites to find software defects. In this study, the experiment method in software engineering was used to analyze the coverage of program modules and program branches. Based on the Defects4J dataset, Evosuite and Randoop tools were used to generate test cases under different generating time limits. When the generation time exceeded 20 s, the numbers of test cases produced by Randoop was more than that produced by Evosuite, but the coverage of Evosuite test cases was significantly higher than the coverage of Randoop. At the same time, this study also analyzed the factors affecting the coverage. It is a good reference for how to use these tools to generate high coverage test case and for the improvement of tools.

Key words: automated unit testing; test cases; test coverage; Evosuite; Randoop

1　研究背景及意义

软件测试是发现软件缺陷的过程, 是保障软件质量的重要手段. 由于被测系统的复杂性以及测试成本的限制, 软件测试自动化已经成为必然. 随着测试技术的进一步发展, 测试流程的不断规范, 大量的单元测试用例生成工具涌现. 这些工具以覆盖率作为测试标准. 有研究者发现, 测试用例的覆盖率是越高, 捕获代码缺陷的可能性就越大. 研究自动化单元测试用例生成工具生成的测试用例的覆盖率和检错率具有重要意义, 直接影响着这些测试工具是否能在业界发挥更好的作用. 在国际上的一些单元测试工具竞赛中[1–6], Evosuite多次获得第一名; 在 2019 年的竞赛中, Randoop 工具和手工编写的测试用例被作为其余工具对比的基线. 因此, 本文选择 Evosuite, Randoop 两个具有代表性的工具为例, 采用了 2016 年单元测试工具竞赛中使用的数据集: Defects4J, 研究自动化单元测试工具的性能, 拟回答以下问题:

① 收稿时间: 2019-12-11; 修改时间: 2020-01-03; 采用时间: 2020-01-14; csa 在线出版时间: 2020-09-04

(1) 在不同的生成时间下, Evosuite, Randoop 生成的测试用例在数量上、方法覆盖率、分支覆盖率等方面的表现如何?

(2) 哪些因素影响了 Evosuite、Randoop 工具生成的测试用例的覆盖率?

2 相关研究

为了分析自动化测试用例生成工具的性能, Fraser 等进行了手工测试和自动化测试的对比实验, 邀请 49 名参与者, 从多个方面分析了手工测试用例和自动化工具 Evosuite 所生成的测试用例的特定[7]. Shamshiri 等在 Defects4J 数据集上研究了 Evosuite、Randoop 工具, 以及商用工具 AgitarOne 的性能, 回答了如何使自动化单元测试工具能够查找到更多的缺陷的问题[8]. Almasi 等使用工业界的项目——LifeCalc 对 Evosuite 和 Randoop 工具的有效性进行了评估[9]. Kotelyanskii 等对基于搜索的单元测试用例工具 Evosuite 的参数调优进行了研究, 其研究结果为本文提供了参考[10]. 在这些研究中, 主要关注的是工具的缺陷查找能力, 但作为自动化单元测试, 更重要的是其对代码的覆盖能力, David Schuler 确定了覆盖率是判断测试用例质量的一个可靠指标, 覆盖率越高, 缺陷查找率可能越高[11]. 本文详细分析了工具生成的测试用例对源代码的方法和分支的覆盖率, 进一步探索了工具的特点和存在的缺陷.

3 实验方法

为了回答上述研究问题, 本文设计了比较实验, 实验过程如图 1 所示. 应用自动化单元测试用例生成工具 Evosuite 和 Randoop, 在 Defects4J 数据集上, 针对不同的被测软件, 分别生成测试用例. 工具生成测试用例的时间分别设置为 10 s, 20 s, 30 s, 60 s, 120 s, 180 s, 300 s. 由于 Evosuite 和 Randoop 都是随机生成测试用例, 每次产生的结果有一定随机性. 因此, 在实验设计上, 同一生成时间重复运行 Evosuite 和 Randoop 3 次, 并以 3 次运行结果的平均值作为实验分析数据.

3.1 Randoop 和 Evosuite 工具

Randoop[12] 是基于反馈的面向对象单元测试的随机测试用例生成工具, 对于被测类, Randoop 创建一系列方法调用和构造函数, 依次创建和更改对象状态. 执行所有创建的序列, 并用执行的生成来创建捕获系统行为的断言. Randoop 的输入是一组要测试的 Java 类, 一个时间限制和一组可选的规范检查器, 生成是一组对应的 JUnit 测试用例. Randoop 的随机种子在缺省情况下为 0, 对于相同被测类, 每次运行时需更改随机种子的值. 另外, Randoop 工具在生成过程中易产生片状测试, 导致 Randoop 工具无法生成更多的测试用例. 为了解决该问题, 需将 flaky-test-behavior 参数设置为 OUTPUT.

Evosuite[13] 是基于搜索的单元测试用例生成工具, 该工具被集成于 Eclipse, IntelliJ, Maven 等环境下, 可以生成 Java 类的 JUnit 测试用例. 该工具采用遗传算法, 从一组随机测试用例开始, 迭代地应用选择、突变和交叉等搜索操作符来进化测试用例, 进化由基于覆盖标准的适应度函数指导, 一旦搜索结束, 测试用例就会被最小化, 并添加测试断言. 在本实验中, 除了禁止 Evosuite 使用单独的类加载器, 更改生成时间以外, 其余均采用默认配置.

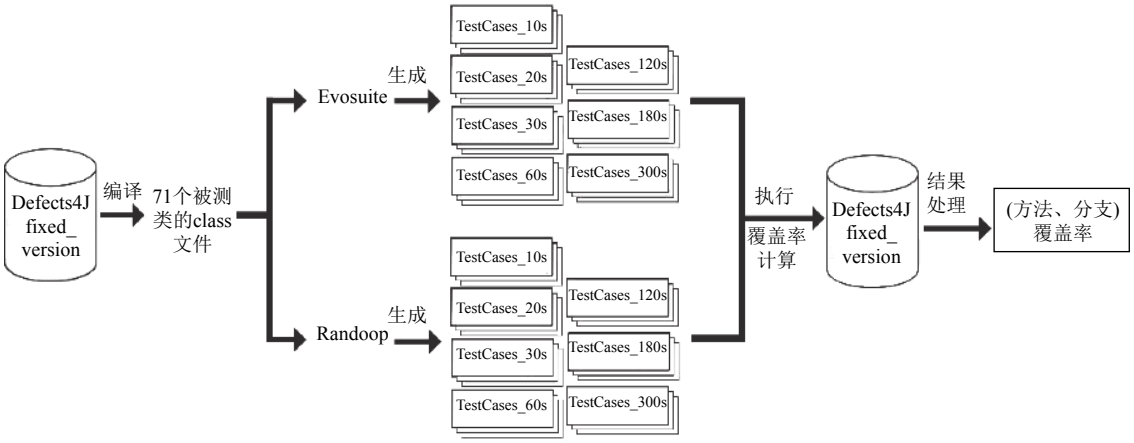

图 1 实验过程设计

3.2 被测软件

实验被测软件采用 Defects4j 数据集[14] 中的 5 个开源项目的多个版本. 这些项目版本中有 357 个真实缺陷: JFreeChart (26 个缺陷)、Google Closure compiler (133 个缺陷)、Apache Commons Lang (65 个缺陷)、Apache Commons Math (106 个缺陷) 和 Joda Time (27 个缺陷). Apache Commons Lang 是处理 Java 基本类方法的工具类包, 弥补了 Java.Lang API 基本处理方法上的不足; Apache Commons Math 是轻量级自容器的数学和统计计算方法类包, 包含大多数常用的数值算法, 外部依赖较少. JFreeChart 是为 Applications, Applets, Servlets 以及 JSP 等使用所设计的, 可生成多种图表, 产生 PNG 和 JPEG 格式的生成. 对于项目版本的每一个缺陷, Defects4j 提供了: (1) 该缺陷对应的缺陷版本和已修复版本; (2) 开发人员手工编写的可以揭露缺陷的测试用例; (3) 缺陷类的列表.

本实验选择 JFreeChart, Apache Commons Lang, Apache Commons Math 作为被测软件, 既满足了多样性的需求, 也保证了所选择的软件的代表性. 实验将 3 个软件每个版本中已修复缺陷的类作为被测类. 当某个类在缺陷类列表中多次出现时, 使用该类第一次出现缺陷时对应的已修复版本. 本实验共筛选出 71 个被测类.

3.3 实验环境

本实验在 Windows 64 位操作系统下采用 SUN Java 8 SE, Eclipse Europa IDE 编译被测项目. 在 JUnit 4 框架下运行测试用例, 使用 EclEmma 插件计算测试用例覆盖率. 编程自动统计覆盖率结果, 并采用 Excel 工具对结果进行分析.

3.4 实验步骤

(1) 编译被测类

Apache Commons Lang 和 Apache Commons Math 是 Maven 构建的项目, 编译该项目时, 需确保 Eclipse 中已配置 Maven 环境.

(2) 测试用例生成

在命令行中使用 Evosuite-1.0.5.jar 和 Randoop-4.0.4 生成测试用例. 在生成测试用例时, 生成时间设为: 10 s, 20 s, 30 s, 60 s, 120 s, 180 s, 300 s. Evosuite 和 Randoop 工具分别在每个生成时间下运行 3 次. 每次运行时, Evosuite 参数保持不变. Randoop 随机种子分别设置为 123, 234, 456, 其余参数保持不变.

(3) 测试用例执行与覆盖率计算

测试用例在 Eclipse 中使用 JUnit 4 框架分别执行. 测试覆盖率由 eclEMMA 插件统计, 结果在测试用例执行结束后得到. Randoop 生成的测试用例分为回归测试用例和揭错测试用例, 这两种测试用例都应被执行. 考虑测试的稳定性, 若测试用例不能执行或执行中断, 则丢弃该测试用例.

本实验从以下几个角度统计覆盖率结果:

① 以类为单位, 统计 Evosuite 和 Randoop 测试用例对被测类的方法覆盖率和分支覆盖率.

② 以方法为单位, 统计 Evosuite 和 Randoop 测试用例在每个方法中的分支覆盖率.

(4) 实验结果处理

实验中存在无法生成测试用例或测试用例无法正常运行的情况, 比如个别类或者方法不存在分支, 此时默认分支覆盖率为 100%.

4 实验结果及分析

4.1 Evosuite 和 Randoop 生成测试用例的数量、方法覆盖率、分支覆盖率

(1) Evosuite 和 Randoop 工具生成的测试用例数量.

图 2 展示了 Evosuite 和 Randoop 工具生成的测试用例数量与生成时间的关系. 无论生成时间如何设置, Evosuite 工具生成的测试用例数量相对稳定. Randoop 工具生成的测试用例数量随着生成时间的增加而增加. 当生成时间为 60 s 时, 明显看出 Randoop 测试用例数量多于 Evosuite. 图 3 展示了在不同项目中 Evosuite 和 Randoop 生成的测试用例数量对比, 当生成时间为 10 s, 20 s 时, Randoop 生成的测试用例数量依旧高于 Evosuite. 因此在所有生成时间设置下, Randoop 工具生成的测试用例数量都多于 Evosuite. 因为 Evosuite 工具会对生成的测试用例进行最小化操作, 保留符合覆盖率标准的测试用例. 而 Randoop 工具则输出生成的所有测试用例.

(2) Evosuite 和 Randoop 工具生成的测试用例的方法覆盖率.

图 4 展示了 Evosuite 和 Randoop 工具生成测试用例对被测类的方法覆盖率与分支覆盖率. 在 3 个被测软件中, Evosuite 测试用例的方法覆盖率随着生成时间的增加而增加. Randoop 测试用例的方法覆盖率在生成时间为 10~120 s 时存在递增现象, 并在 120 s 时达到

最大值. 在 Chart 项目中, 当生成时间为 300 s 时, Randoop 测试用例的方法覆盖率有所降低; 在 Lang 项目上, 当生成时间超过 120 s 后, Randoop 测试用例的方法覆盖率保持平稳; 对于 Math 项目, 当生成时间超过 120 s 后, Randoop 测试用例的方法覆盖率逐渐降低. 当且仅当生成时间为 10 s 时, 对于 Chart 项目, Randoop 测试用例的方法覆盖率高于 Evosuite. 当生成时间达到 20 s 后, 无论在哪个项目上, Evosuite 测试用例的方法覆盖率都高于 Randoop. 总体来看, Evosuite 测试用例的方法覆盖率高于 Randoop.

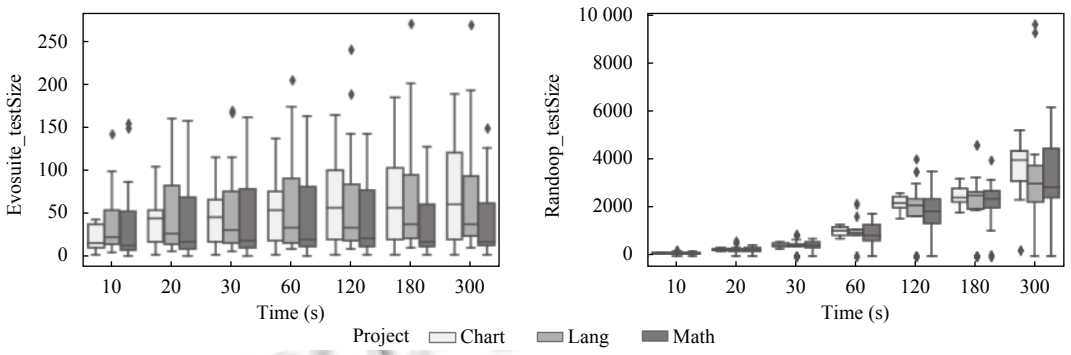

图 2 Evosuite 和 Randoop 工具生成的测试用例数量与生成时间的关系

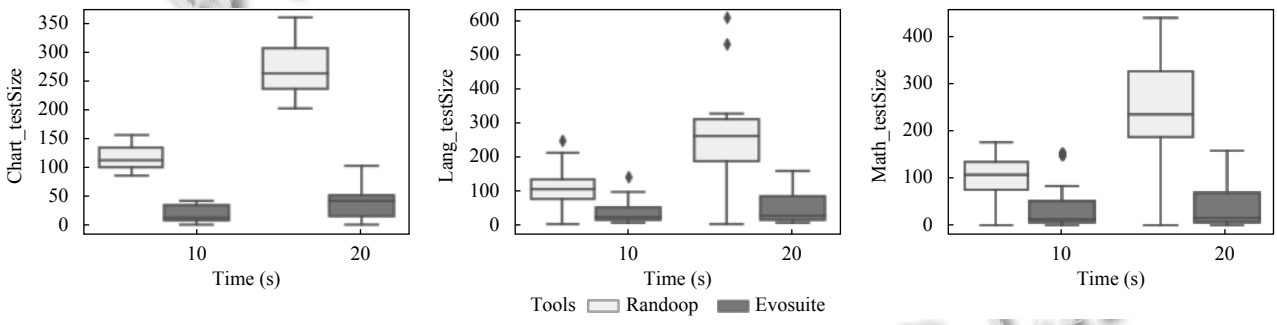

图 3 在不同的项目中, Evosuite 和 Randoop 生成的测试用例数量对比

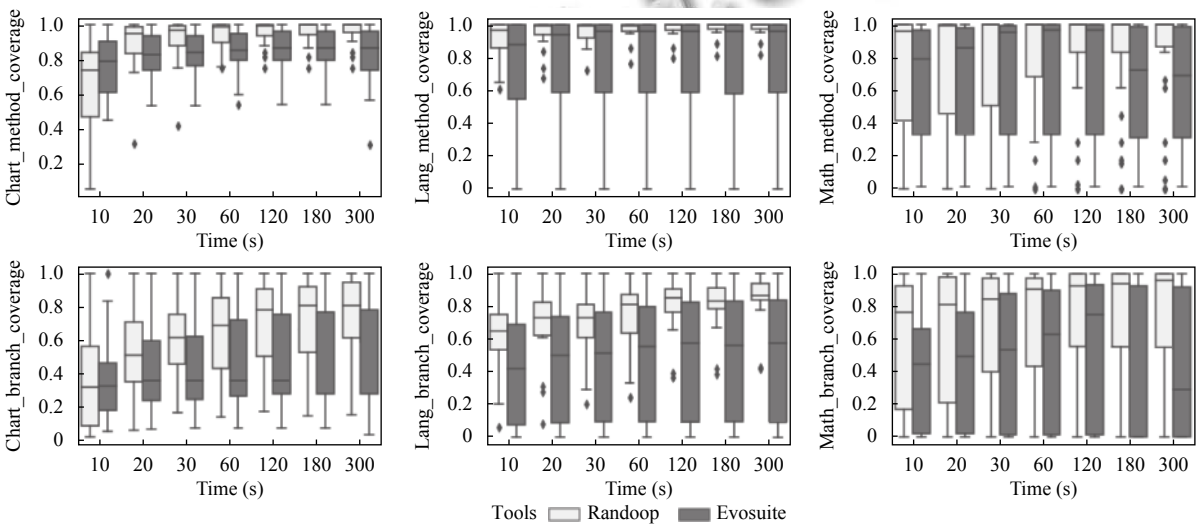

图 4 Evosuite 和 Randoop 测试用例对被测类的方法覆盖率与分支覆盖率比较

(3) EvosuiteRandoop 工具生成的测试用对被测类的分支覆盖率.

在 Chart 和 Math 项目中, Evosuite 工具生成的测试用例对被测类的分支覆盖率随着生成时间的增加逐渐上升. 在 Lang 项目中, 当生成时间为 30 s 时, Evosuite 工具生成的测试用例的分支覆盖率较生成时间比 20 s 时略有降低. 但在其余生成时间上, Evosuite 工具生成的测试用例的分支覆盖率依旧随着生成时间的增加而增加. 当生成时间为 10~120 s 时, Randoop 工具生成的测试用例的分支覆盖率随着生成时间的增加而增加. 当生成时间达到 120 s 后, 在 Chart 和 Lang 项目上, Randoop 工具生成的测试用例的分支覆盖率达到最优值, 并保持稳定. 在 Math 项目上, Randoop 工具生成的测试用例的分支覆盖率逐渐下降. 当且仅当生成时间为 10 s 时, 在 Chart 项目上 Randoop 工具生成的测试用例分支覆盖率与 Evosuite 工具生成的测试用例分支覆盖率相当, 当生成时间达到 20 s 后, Evosuite 测试用例的分支覆盖率高于 Randoop 测试用例. 整体来看, Evosuite 对被测类的分支覆盖率同样优于 Randoop.

(4) 影响 Evosuite 对被测类的分支覆盖率高于 Randoop 的因素.

从上述结论可知, Evosuite 对被测类方法覆盖率高于 Randoop. 原因之一是 Randoop 工具的随机性, 不能覆盖部分方法, 就不能覆盖到方法中的分支, 而 Evosuite 可实现更多的方法覆盖, 可覆盖到方法中的分支. 图 5 展示了同时被 Evosuite 和 Randoop 测试用例覆盖的方法的分支覆盖率. 从图 5 可见, 当 Evosuite 和 Randoop 同时覆盖某个方法时, Evosuite 测试用例对该方法的分支覆盖程度高于 Randoop. 因此, Evosuite 对被测类分支覆盖率高于 Randoop 不仅因为 Evosuite 覆盖了更多的方法, 还因为 Evosuite 测试用例较 Randoop 可以覆盖到方法中的更多分支.

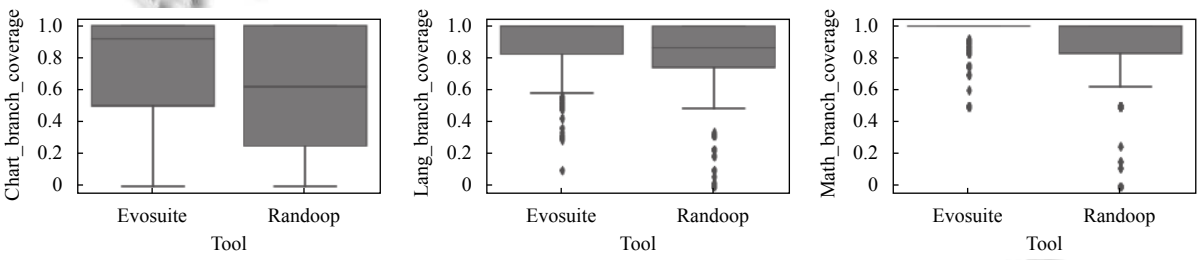

图 5 同时被 Evosuite 和 Randoop 测试用例覆盖的方法的分支覆盖率

4.2 影响 Evosuite 和 Randoop 工具生成的测试用例覆盖率的因素

通过分析被测软件, 发现软件版本中存在抽象类, 而 Randoop 工具在执行时, 会忽略这些抽象类以及接口. 所以, 即使抽象类中存在着具体的方法 (例如: Math_12, Lang_15 等), Randoop 工具也没有对这些方法生成测试用例. 可以认定, 被测软件中存在的抽象类是影响 Randoop 工具生成的测试用例覆盖率的因素之一. 在 Randoop 工具改进中, 建议考虑对抽象类的处理.

图 6 展示了 Randoop 工具所生成的一个测试用例的样例. 在执行该用例时, 发现深色标注的代码未执行, 从而未覆盖该代码所调用的被测方法. 从这一测试用例中可以看出, Randoop 在生成测试用例时, 随机模拟被测方法中所需要的参数来调用方法. 当 Randoop 工具不确定测试代码是否能成功执行时, 将会以 try-catch 语句包裹该代码. 实验证明, 若测试用例中某些语句被 try-catch 包裹, 则该语句基本无法正常执行, 从而导致无法覆盖其中被调用的被测方法. 因为 Randoop 不能成功模拟调用该被测方法所需要的参数. Randoop 的设计者也提到:目前 Randoop 对于参数的选择是随机的. 因此, 有效模拟被测方法所需参数是影响 Randoop 覆盖率的因素之一. 当 Randoop 需要模拟某些参数来调用方法时, 可以依照上下文关系获取该参数的值, 而不是重新定义随机值[15].

在图 4 中, 当生成时间足够长时, Evosuite 测试用例的方法覆盖率达到 90% 以上, 方法覆盖程度并不是影响该工具分支覆盖率的重要因素. 图 7 展示了 Evosuite 工具对某被测模块的覆盖情况的样例, Evosuite 工具所生成的测试用例虽然覆盖了该方法, 但当方法中出现分支时, Evosuite 无法覆盖该分支中所包含的语句, 该分支的运行依赖于其他方法的执行结果. 因此, Evosuite

工具对被测方法相关的依赖方法结果的构造是影响分支覆盖的因素之一.

```
public void test005() throws Throwable {
    if (debug)
        System.out.format("%n%s%n", "RegressionTest0.test005");
    org.apache.commons.math3.linear.OpenMapRealVector openMapRealVector2 = new
org.apache.commons.math3.linear.OpenMapRealVector((int) (byte) 1, (-1));
    java.lang.Double[] doubleArray6 = new java.lang.Double[] { 1.0d, 0.0d, (-1.0d) };
    org.apache.commons.math3.linear.OpenMapRealVector openMapRealVector8 = new
org.apache.commons.math3.linear.OpenMapRealVector(doubleArray6, (double) 'a');
    try {
        double double9 = openMapRealVector2.getL1Distance((org.apache.commons.math3.linear.RealVector)
openMapRealVector8);
        org.junit.Assert.fail("Expected exception of type
org.apache.commons.math3.exception.DimensionMismatchException; message: 1 != 3");
    } catch (org.apache.commons.math3.exception.DimensionMismatchException e) {
    }
    org.junit.Assert.assertNotNull(doubleArray6);
}
```

图 6 Randoop 工具生成的一个测试用例样例

```
public void drawDomainTickBands(Graphics2D g2, Rectangle2D dataArea,
                                List ticks) {
    Paint bandPaint = getDomainTickBandPaint();
    if (bandPaint != null) {
        boolean fillBand = false;
        ValueAxis xAxis = getDomainAxis();
        double previous = xAxis.getLowerBound();
        Iterator iterator = ticks.iterator();
        while (iterator.hasNext()) {
            ValueTick tick = (ValueTick) iterator.next();
            double current = tick.getValue();
            if (fillBand) {
                getRenderer().fillDomainGridBand(g2, this, xAxis, dataArea,
                        previous, current);
            }
            previous = current;
            fillBand = !fillBand;
        }
        double end = xAxis.getUpperBound();
        if (fillBand) {
            getRenderer().fillDomainGridBand(g2, this, xAxis, dataArea,
                    previous, end);
        }
    }
}
```

图 7 Evosuite 工具对某被测模块的覆盖情况的样例

4.3 有效性分析

内部有效性, Evosuite 和 Randoop 工具都具有一定的随机性, 若要评估工具测试用例的覆盖率, 应生成尽可能多的测试用例. 本实验在多个项目、多个版本、多个生成时间上分别生成 3 组测试用例, 实验数据具有一定的统计意义. 在实验过程中, Randoop 工具每次运行所生成的测试用例的方法覆盖率和分支覆盖率波动较小; Evosuite 覆盖率波动情况较 Randoop 更大, 若以工具的最优值作为衡量标准, Evosuite 测试用例的覆盖率结果将更优. 但就 Evosuite 运行情况来看,采用最小值、最大值、平均值完整体现工具的真实情况, 衡量工具覆盖率更为合理.

外部有效性, 从实验结果来看, Evosuite 工具生成的测试用例的方法覆盖率和分支覆盖率随着生成时间的增加而增加. Randoop 工具在生成时间达到 120 s 后, 对于不同的项目方法覆盖率和分支覆盖率具有不同的特征. 但 Lang 和 Math 项目是 Apache Commons 中的基础工具包, 使用频率较高, Chart 项目充分体现了绘图程序的基本特征, 因此, 本实验选择的被测项目既满足了多样性的需求, 也保证了所选择的软件的代表性.

5 结论

本实验通过对 Evosuite 和 Randoop 工具所生成的测试用例的覆盖率分析发现: ① 即使生成时间低至 10 s, Radnoop 所生成的测试用例数量依旧高于 Evosuite. 并且当生成时间到达 20 s 后, 单元测试工具 Evosuite 生成的测试用例的方法覆盖率和分支覆盖率覆盖率高于 Randoop. ② 影响 Randoop 工具覆盖率的原因包含如何处理抽象类以及如何模拟被测方法所需要的参数等. Evosuite 工具提高覆盖率的关键问题是如何构造与被测方法相关的依赖方法结果, 以保证能够覆盖更多的分支. 在未来的工作中, 可以研究是否可以采用符号化执行与随机测试相结合的方法提高 Randoop 工具的参数模拟能力.

参考文献

1 Fraser G, Arcuri A. Evosuite at the SBST 2013 tool competition. Proceedings of 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops. Luxembourg City, Luxembourg. 2013. 406–409.

2 Fraser G, Arcuri A. Evosuite at the SBST 2015 tool competition. Proceedings of 2015 IEEE/ACM 8th International Workshop on Search-Based Software Testing. Florence, Italy. 2015. 25–27.

3 Fraser G, Arcuri A. Evosuite at the SBST 2016 tool competition. Proceedings of 2016 IEEE/ACM 9th International Workshop on Search-Based Software Testing (SBST). Austin, TX, USA. 2016. 33–36.

4 Fraser G, Rojas JM, Campos J, *et al*. Evosuite at the SBST 2017 tool competition. Proceedings of 2017 IEEE/ACM 10th International Workshop on Search-Based Software Testing (SBST). Buenos Aires, Argentina. 2017. 39–42.

5 Fraser G, Rojas JM, Arcuri A. Evosuite at the SBST 2018 tool competition. Proceedings of 2018 IEEE/ACM 11th International Workshop on Search-Based Software Testing (SBST). Gothenburg, Sweden. 2018. 34–37.

6 Campos J, Panichella A, Fraser G. Evosuite at the SBST 2019 tool competition. Proceedings of 2019 IEEE/ACM 12th International Workshop on Search-Based Software Testing (SBST). Motreal, QC, Canada. 2019. 29–32.

7 Fraser G, McMinn P, Staats M, *et al*. Does automated white-box test generation really help software testers? Proceedings

of the 2013 International Symposium on Software Testing and Analysis. New York, NY, USA. 2013. 291–301.

8 Shamshiri S, Just R, Rojas JM, *et al*. Do automatically generated unit tests find real faults? An empirical study of effectiveness and challenges (T). Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering. Clayton, Australia. 2015. 201–211.

9 Almasi MM, Hemmati H, Fraser G, *et al*. An industrial evaluation of unit test generation: Finding real faults in a financial application. Proceedings of 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP). Buenos Aires, Argentina. 2017. 263–272.

10 Kotelyanskii A, Kapfhammer GM. Parameter tuning for search-based test-data generation revisited: Support for previous results. Proceedings of 2014 14th International Conference on Quality Software. Dallas, TX, USA. 2014. 79–84.

11 Schuler D, Zeller A. Assessing oracle quality with checked coverage. Proceedings of 2011 Fourth IEEE International Conference on Software Testing, Verification and Validation. Berlin, Germany. 2011. 90–99.

12 Pacheco C, Ernst MD. Randoop: Feedback-directed random testing for Java. Companion to the 22nd ACM SIGPLAN Conference on Object-Oriented Programming Systems and Applications Companion. New York, NY, USA. 2007. 815–816.

13 Fraser G, Arcuri A. Whole test suite generation. IEEE Transactions on Software Engineering, 2013, 39(2): 276–291. [doi: 10.1109/TSE.2012.14]

14 Just R, Jalali D, Ernst MD. Defects4J: A database of existing faults to enable controlled testing studies for java programs. Proceedings of the 2014 International Symposium on Software Testing and Analysis. New York, NY, USA. 2014. 437–440.

15 Randoop project ideas. https://randoop.github.io/randoop/projectideas.html. [2020-01-09].