计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2022,31(10):1−14 [doi: 10.15888/j.cnki.csa.008680]

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

模糊测试改进技术评估①

张　阳[1,2], 佟思明[1,2], 程　亮[1,2], 孙晓山[1,2]

[1](中国科学院大学, 北京 100049)
[2](中国科学院 软件研究所 可信计算与信息保障实验室, 北京 100190)
通信作者: 佟思明, E-mail: siming2019@iscas.ac.cn

摘　要: 模糊测试技术在发现真实程序漏洞中具有突出效果. 近年来, 模糊测试改进技术受到了相关学者的广泛关注, 大量的优化模糊测试工具被相继提出, 被提出的优化模糊测试工具多数结合了多种改进技术以期望达到更好的效果. 然而, 当前仍然缺乏对单一模糊测试改进技术的系统性评估与分析. 本文首先基于 4 个指标, 设计建立了一个针对单一模糊测试改进技术的评估体系, 然后基于所提出的评估体系, 对近年提出的先进模糊测试工具中集成的多个单一模糊测试改进算法进行了多组实验以评估不同改进技术类别中各个单一改进技术的改进效果, 并结合实验数据与实际算法设计和代码实现进行分析. 我们希望通过对单一模糊测试改进技术的评估与分析能够对未来的模糊测试改进研究工作提供帮助.

关键词: 漏洞挖掘; 模糊测试; 评估; 改进技术; 信息安全

引用格式: 张阳,佟思明,程亮,孙晓山.模糊测试改进技术评估.计算机系统应用,2022,31(10):1–14. http://www.c-s-a.org.cn/1003-3254/8680.html

Evaluation of Fuzzing Improving Techniques

ZHANG Yang[1,2], TONG Si-Ming[1,2], CHENG Liang[1,2], SUN Xiao-Shan[1,2]

[1](University of Chinese Academy of Sciences, Beijing 100049, China)
[2](Trusted Computing and Information Assurance Laboratory, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)

Abstract: Fuzzing is outstanding in detecting vulnerabilities in real-world programs. In recent years, researchers have paid considerable attention to fuzzing improving techniques, and large numbers of optimized fuzzers were proposed. These fuzzers are usually combinations of more than one improving technique for better performance. However, systematic evaluation of individual fuzzing improving techniques is still to be conducted. In this study, we establish an evaluation system for such techniques according to four metrics and used it to evaluate individual fuzzing improving algorithms integrated into recently proposed advanced fuzzers. Multiple groups of experiments are conducted to evaluate the effectiveness of each individual technique in each category of improving techniques, and the experimental data are comprehensively analyzed with the actual algorithm design and code implementation. We hope the evaluation of individual fuzzing improving techniques could help researchers develop more effective fuzzers in the future.

Key words: vulnerability detection; fuzzing; evaluation; improving techniques; information security

1　绪论

近年来, 利用软件漏洞实施安全攻击的行为层出不穷, 如何更好地保证软件质量和安全, 减少软件及系统漏洞, 成为当前安全领域一个重要的研究方向.

模糊测试[1]是一种动态软件漏洞挖掘技术, 能够有效检测软件或计算机系统中的安全漏洞. 其核心思

① 基金项目: 国家自然科学基金 (62072448)
收稿时间: 2021-12-15; 修改时间: 2022-01-13; 采用时间: 2022-01-26; csa 在线出版时间: 2022-06-16

想是将变异生成的随机数据输入待测试程序中, 监视程序运行情况, 当发现程序出现崩溃或其他异常情况时保存数据, 从而找到程序漏洞. 相比于其他软件安全检测技术, 如代码审计[2]、污点分析[3]、符号执行[4] 等, 模糊测试技术具有自动化程度高, 时间开销小, 应用范围广的优点, 模糊测试作为一种高效的漏洞挖掘技术在近些年得到了学术界及工业界的广泛关注和应用.

模糊测试逐渐成为最有效的漏洞挖掘技术之一, 但受其动态测试的本质所限, 模糊测试的代码覆盖率难以达到较高水平. 为了提高模糊测试技术的效率与准确率, 近年来, 相关领域学者在模糊测试技术上提出了一系列改进策略与算法[1]. 这些工作通常对模糊测试过程中的多个步骤同时进行了改进, 包含多种改进技术, 以追求更好的改进效果. 然而, 当前仍没有工作对单一改进技术的改进效果及其原因进行系统全面的评估和分析. 因此, 本文建立了单一模糊测试改进技术评估体系, 对近年来信息安全高水平会议期刊模糊测试改进工具所包含的单一改进技术进行拆分、评估与分析, 基于改进算法特点进行分类, 通过数据对比得到不同改进类别中改进算法的实际效果评价, 希望对模糊测试改进技术有更清晰深入的认识, 同时根据不同改进技术的不同改进效果原因分析, 帮助未来的模糊测试改进工作取得更好的效果.

2 研究背景

本节首先介绍模糊测试技术的基本分类, 说明基于变异的灰盒模糊测试工具工作流程, 然后介绍经典基于变异的灰盒模糊测试工具 American Fuzzy Lop (AFL)[5].

2.1 模糊测试技术分类

当前模糊测试技术方法根据样本生成方法可以分为基于变异的模糊测试以及基于生成的模糊测试两类[6].

(1) 基于变异的模糊测试

基于变异的模糊测试通过改变已有的数据样本去生成测试数据. 对于种子样本, 模糊测试工具通过对该样本进行位翻转、数学加减、插入删除、拼接等操作, 得到新的种子, 遍历更多的程序执行路径, 以达到更高的代码覆盖, 从而发现更多程序中可能存在的漏洞.

(2) 基于生成的模糊测试

基于生成的模糊测试则主要应用于针对协议、JS 引擎等以高度结构化格式文件为输入的目标程序, 由于这些程序中存在大量的格式检查, 输入文件的不同字段具有一定的语法语义, 随机的变异往往无法满足要求, 因此需要通过一定的规则和算法生成输入文件.

根据对待测试程序内部结构的认知程度, 模糊测试可以分为白盒模糊测试、黑盒模糊测试以及灰盒模糊测试 3 类[6].

(1) 黑盒模糊测试

黑盒模糊测试不考虑待测试程序的内部逻辑, 持续生成随机样本输入程序, 观察待测试程序的执行情况.

(2) 白盒模糊测试

白盒模糊测试指在获取到待测试程序的内部实现信息, 如源代码、设计细节以及运行时信息等之后, 利用已有信息针对性地对程序进行测试的方法.

(3) 灰盒模糊测试

灰盒模糊测试介于黑盒模糊测试和白盒模糊测试之间, 通过获取一些待测试程序的信息, 如运行时分支覆盖来指导模糊测试从而提高测试效率. 常见灰盒模糊测试工具通过轻量级代码插桩等方法获取所需信息.

基于变异的灰盒模糊测试具有应用范围广、执行效率高的特点, 受到了相关学者的广泛关注, 近年提出的模糊测试工作多数基于此类型, 因此本文主要针对基于变异的灰盒模糊测试改进技术进行评估与分析.

2.2 基于变异的灰盒模糊测试工具流程

多数基于变异的灰盒模糊测试工具工作流程如图 1 所示, 首先从初始种子队列中选取合适的输入, 然后, 模糊测试工具重复地对这些输入进行变异并执行待测试程序. 如果程序运行中出现了异常行为, 则模糊测试工具保留变异的输入以便将来使用, 并记录观察到的具体信息. 最终, 由于达到特定目标, 如发现某种漏洞, 或用户主动停止, 模糊测试结束.

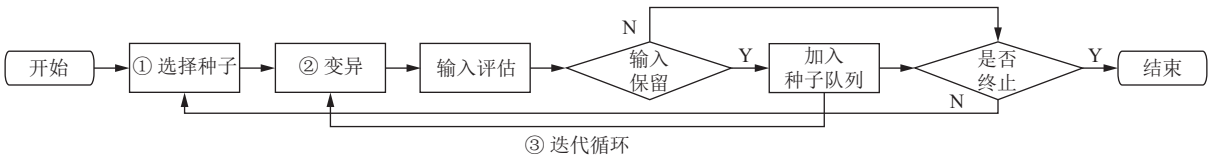

图 1 基于变异的灰盒模糊测试工具流程图

2.3 American Fuzzy Lop (AFL)

AFL[5] 由 Google 的 Michal Zalewski 所开发, 是当前最流行的模糊测试工具之一.

AFL 是一个采用遗传算法基于变异的灰盒模糊测试工具, 能够有效提高待测试程序的代码覆盖率. AFL 需要用户提供待测试程序、运行测试应用程序的示例命令和至少一个小的示例输入文件. 然后, AFL 尝试实际执行指定的命令, 通过对输入文件进行各种变异修改来执行待测试程序的不同路径. 当测试程序崩溃或挂起时, 可能意味着发现了一个新的安全漏洞. 在这种情况下, AFL 将保存修改后的输入文件以供用户进一步检查. 为了最大限度地提高模糊测试性能, AFL 在编译过程中通过代码插桩, 在程序运行时记录分支覆盖情况, 进一步引导模糊测试. 模糊测试工具 AFL 的具体工作流程如图 2 所示.

AFL 作为经典的基于变异的灰盒模糊测试工具, 设计结构完整, 算法准确高效, 可扩展性强, 近年来很多模糊测试改进工作选择基于 AFL 实现[7−11], 因此本文选择 AFL 作为基准模糊测试工具进行改进技术的评估与分析.

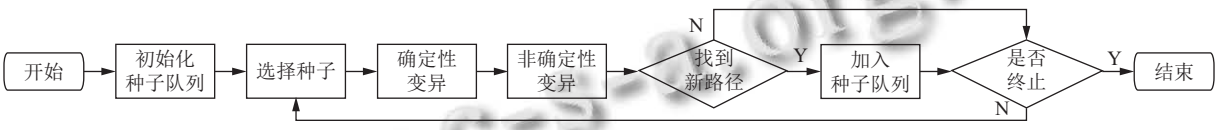

图 2 AFL 工作流程图

3 模糊测试改进技术总结

本节首先介绍对模糊测试改进技术的分类以及相应类别中近年提出的先进改进技术, 然后对这些单一改进技术进行整理总结与分析, 如图 3 所示.

3.1 模糊测试改进技术分类与介绍

由第 2 节介绍可知, 当前基于变异的灰盒模糊测试工具工作流程可根据不同操作进行较为清晰的阶段划分. 基于不同模糊测试阶段的操作复杂程度与近年相关学者对不同阶段的改进关注程度, 本文将模糊测试改进技术分为了 4 个类别, 即种子选取策略改进、变异策略改进、能量分配策略改进以及其他方面的改进技术. 下面将对各类别的改进技术进行说明, 并对属于不同类别的一些改进技术进行简单介绍.

(1) 种子选取策略改进技术

该类别的改进技术重点解决如何从种子队列中选择高质量种子进行变异的问题. AFLFast[12] 倾向于选择低频路径更多的、更少被选择的种子进行变异; FairFuzz[13] 提出了低频边的概念, 在测试过程中记录不同边被访问的次数, 仅选择具有低频边的种子进行进一步变异; CollAFL[7] 优先选择具有更多内存访问操作、未访问的邻居边或未访问的邻居后继的种子; SAFL[8] 通过符号执行生成初始种子集合, 并在测试中优先选择具有稀有边的种子; EcoFuzz[14] 将模糊测试分为了探索 (exploration) 和利用 (exploitation) 两个阶段, 在探索阶段按照顺序选择种子, 利用阶段优先选择自转移频率低且队列中编号较大的种子.

(2) 变异策略改进技术

当从队列中选择了种子后, 模糊测试工具对选择的种子进行一系列变异操作以生成新样本, 访问目标程序中新的路径, 从而触发新的漏洞. 变异策略改进技术通过优化对选定种子的变异操作和变异位择, 提高变异效率, 从而提高模糊测试效果. Angora[11] 通过污点分析技术搜索种子中能够提高代码覆盖率的变异操作和变异位置, 并在模糊测试过程中通过梯度下降算法解决路径约束问题; Böttinger 等人[15] 提出通过强化学习得到能够增加代码覆盖的变异操作及变异位置; FairFuzz[13] 在确定性变异阶段记录不影响低频边覆盖的变异操作和变异位置, 并在后续随机的非确定性变异阶段进行相应变异; MOPT[16] 通过粒子群算法优化变异操作的选择; Karamcheti 等人[17] 提出根据汤普森采样算法选择当前最优的变异操作.

(3) 能量分配策略改进技术

能量分配指, 在非确定性变异阶段对选定种子选择随机变异次数的过程. 能量分配策略改进技术通过对种子的价值进行评估, 优化对不同种子选择的变异次数, 对更有可能生成更多有价值样本的种子赋予更多的变异次数. AFLFast[12] 基于马尔科夫链提出了 5 种不同的能量分配策略, 根据种子的被选择的次数以及访问路径被访问的次数计算分配的变异次数; BanditAFL[18] 将能量分配问题看作多臂老虎机问题, 通过强化学习

计算最优的能量分配策略; EcoFuzz[14] 首先计算发现新路径平均所需的变异次数, 并根据当前种子路径被访问的次数以及上一个种子发现新路径的变异次数进行调整.

(4) 其他方面改进技术

除上述 3 种常见模糊测试改进技术类别外, 一些工具提出了针对模糊测试不同阶段的多种其他方面改进技术. 如, CollAFL[7] 指出传统的边覆盖计算是不精确的, 提出了通过解决哈希碰撞问题计算更准确的边覆盖的方法; INSTRIM[19] 修改了传统 AFL 的插桩方法以达到更快的代码执行速度; Xu 等人[20] 通过优化系统调用等方法提高模糊测试效率. 该类别的改进技术通常从较独特的角度优化模糊测试流程或尝试解决较少被考虑的问题, 因此我们不对这一类别的改进技术进行进一步划分.

3.2 模糊测试改进技术总结与分析

我们对近年来在安全高水平会议期刊中发表的灰盒模糊测试改进工作共 46 篇文章[7,8,11–54] 进行了整理, 将其中的改进技术进行分类. 按照第 3.1 节中的分类标准, 将基于变异的灰盒模糊测试改进技术划分为种子选取策略改进、变异策略改进、能量分配策略改进以及其他方向改进 4 类. 模糊测试相关改进技术整理如图 3 所示. 其中横坐标表示模糊测试改进技术提出的时间, 纵坐标表示改进所属类别, 将每个改进工具根据以上两个标准归纳入不同单元格内, 每个单元格颜色表示包含该类别改进技术的工具数量, 颜色所表示含义参照右侧图例. 其中工具中所标注的*表示该模糊测试工具基于 AFL 实现.

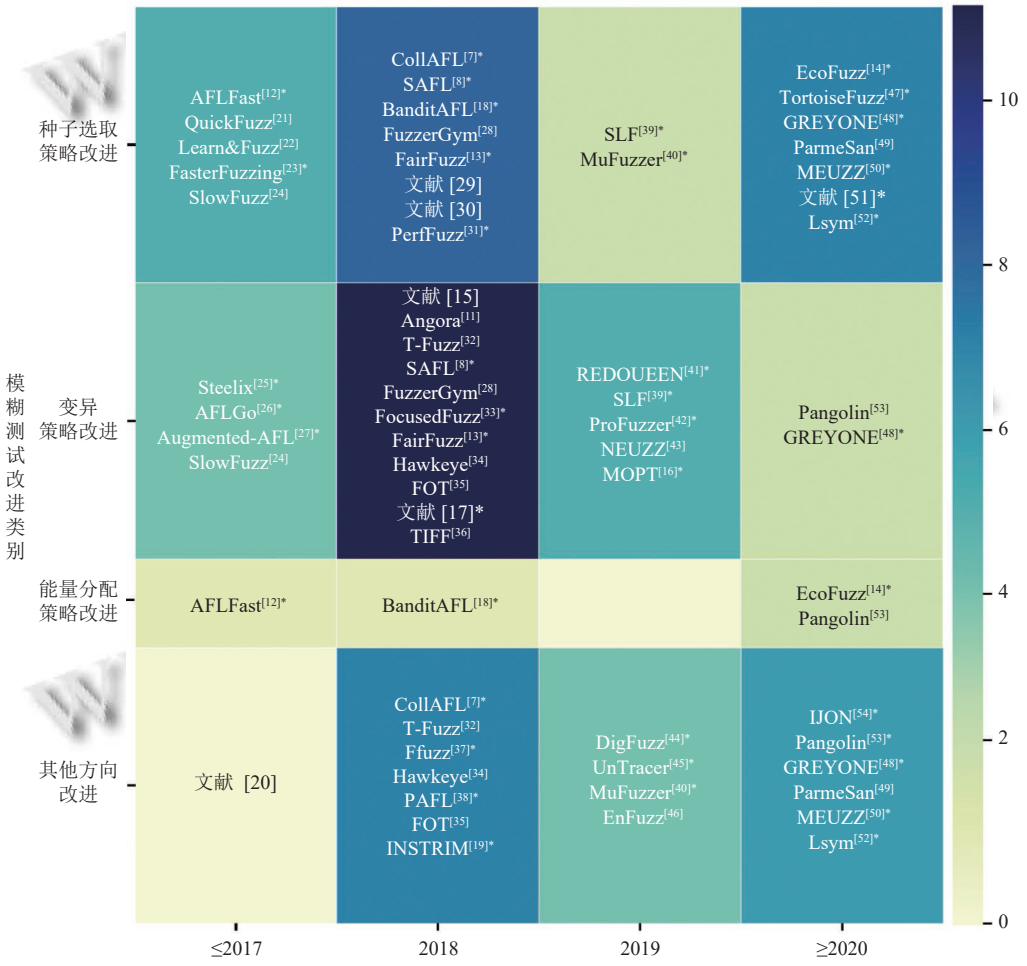

图 3 模糊测试改进技术分类整理

由图 3 分析可知, 从整体上看, 在整理的 46 个模糊测试改进工具中, 47.83% (22/46) 的工具提出了种子选取策略改进, 47.83% (22/46) 的工具提出了变异策略改进, 8.70% (4/46) 提出了能量分配策略改进, 39.13%

(18/46) 提出了其他方向的改进方法. 从改进类别方面分析, 模糊测试研究人员更多关注种子选取和变异策略的改进, 而对能量分配策略以及其他方向的改进关注度较低. 从改进技术所提出的时间分析, 自从 2015 年高效的模糊测试工具 AFL 被提出实现, 模糊测试改进技术进入快速发展阶段, 每年有很多不同类别的改进工作被陆续提出, 2018 年达到一个高峰期, 2019 年后由于模糊测试代码覆盖率的瓶颈问题没有得到有效解决, 且没有更多快速发展的模糊测试新兴方向, 模糊测试改进工作的热度呈下降趋势.

此外, 在 46 个模糊测试改进工具中, 65.22% (30/46) 的工具基于 AFL 改进实现, 且几乎所有工具将 AFL 作为对照工具进行相应改进技术的评估, AFL 在当前的模糊测试改进工作中至今仍处于非常重要的地位.

4 评估体系建立

为了系统量化地评估单一模糊测试改进技术, 本文建立了单一模糊测试改进技术评估体系. 本节首先对评估体系设计进行整体介绍, 然后说明评估体系指标选取, 最后对说明综合评估结果的计算模型.

4.1 评估体系概述

该单一模糊测试改进技术评估体系整体设计如图 4 所示. 首先准备待评估的模糊测试改进技术, 确定基准模糊测试工具, 在其之上结合待评估的模糊测试改进技术, 形成单一改进模糊测试工具. 选择用于测试的程序集, 将单一改进模糊测试工具在测试程序集上进行多次固定时间的模糊测试, 获得队列中的种子样本、崩溃样本, 以及其他模糊测试的运行时信息. 之后对获取到的数据进行处理, 计算得到单一改进模糊测试工具的边覆盖数量、去重后的崩溃样本数量、低频边比例以及内存操作指令数量 4 个指标, 根据算法计算得到最终的模糊测试改进效果评估结果.

4.2 评估体系指标选取

根据近期国内外模糊测试改进工作的整理, 结合对模糊测试改进效果影响因素的分析, 选取了 4 个用于评估的指标, 即测试程序边覆盖率、去重后崩溃样本数量、覆盖低频边所占比例以及覆盖测试程序内存操作指令的数量.

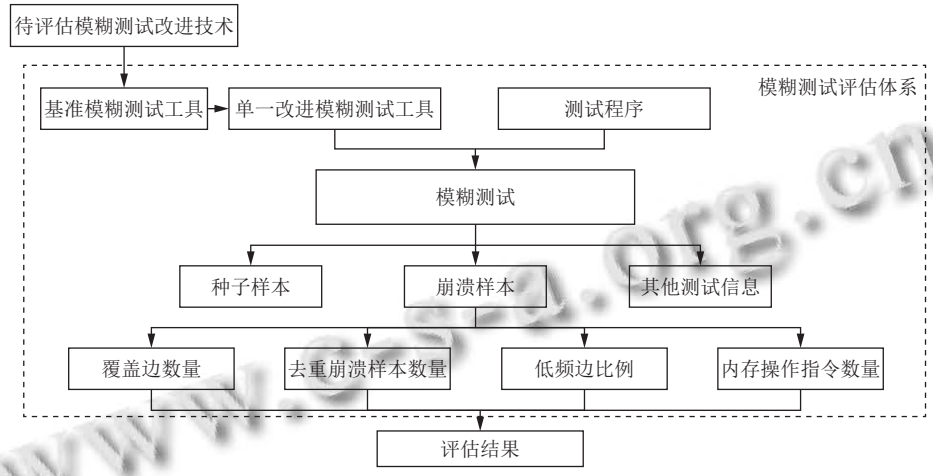

图 4 模糊测试评估体系架构图

(1) 边覆盖数量

测试程序中的边用于反映程序中基本块之间的执行路径关系, 边的覆盖率越高, 测试的程序路径越全面, 从而能够发现其中更多的潜在漏洞. 与路径覆盖相比, 边覆盖指标只关注覆盖边的情况而忽略发现新边的种子以及新边在一次执行中覆盖的次数, 更能清晰准确地反映程序覆盖情况. 近年的模糊测试改进工作评估中往往将边覆盖率作为评价模糊测试工具效果的基本指标, 因此将边覆盖数量作为该评估体系的第 1 个指标.

(2) 去重崩溃样本数量

在模糊测试中, 程序由于异常输入所导致的崩溃, 很可能是因为触发了程序中潜在的漏洞, 当程序在执行过程中发生崩溃, AFL 会首先检查此前的崩溃输入中是否有与当前输入的分支覆盖完全相同的输入样本, 若没有则保存. 通常情况下, 崩溃样本越多, 可能存在的漏洞数量则越多. 然而, AFL 所收集的崩溃样本可能

存在大量重复. 如图 5 所示, 假设此时存在一个待检测的程序代码片段, 其第 5 行的 crash() 函数存在漏洞, 当两个输入的变量 a 分别为 True 和 False 的时候都会触发该漏洞, 由于执行路径不同, AFL 会将两个输入样本均保留, 然而这两个样本实际上触发的为一个漏洞. 因此, 在评估以 AFL 为基础的模糊测试工具时, 将 AFL 原始保留的崩溃样本进行去重, 将调用栈相同的样本看做同一个崩溃样本, 计算去重后的崩溃样本数量, 作为评估体系的第 2 个指标.

```
1: if (a)
2:   res=0;
3: else
4:   res=1;
5: crash( );
```

图 5 触发漏洞示例程序代码片段

(3) 低频边比例

在模糊测试中发现边的数量整体上可以反映模糊测试工具的漏洞挖掘效果, 但已发现边的价值是有差异的. 因此, 所提出模糊测试评估体系通过低频边比例来评价所发现边的价值. 低频边比例指标指, 对于一个测试程序所发现的边中, 访问概率低的边占所发现边总数的比例. 在模糊测试相关研究中, 很多论文强调了低频分支的重要性[12–14,40,55]. 模糊测试工具变异生成的样本往往倾向于多次遍历少量高频边, 而程序中大量的边很少被访问. 因此, 模糊测试工具访问的边中低频边比例越高, 其价值也相对较高, 低频边比例为评估体系的第 3 个指标.

(4) 内存操作指令数量

内存操作指令数量同样可用于评价发现边的价值. 一些模糊测试相关研究论文中指出, 多数程序漏洞为错误的内存访问操作所引起, 如内存越界读写[7,47,49]. 目标程序中的内存访问操作指令越多, 可能隐含的漏洞则越多. 因此, 统计模糊测试工具访问的基本块中所包含的内存操作指令数量, 并将其作为模糊测试评估系统的第 4 个指标.

4.3 综合评估结果计算

为得到多个模糊测试改进工具的效果评价比较结果, 在统计得到多个指标的数据结果后, 根据优序图算法[56] 计算得到不同指标的权重, 通过 technique for order preference by similarity to ideal solution (TOPSIS)[57] 算法计算每个模糊测试工具的效果分数, 并对分数排序得到效果比较结果. TOPSIS 算法常用于解决多指标评价问题, 相较于类似算法, 如灰色关联度分析或模糊综合评价模型, TOPSIS 算法更适用于指标数据完整准确评价排序问题, 因此选择 TOPSIS 算法作为模糊测试效果评估的主要算法. 该模糊测试改进评估体系具体包括以下步骤.

(1) 对评估指标数据进行标准化处理. 由于所提出的 4 个评估指标数量单位不同, 需要通过标准化处理转换为无量纲数据进一步用于评估结果计算. 数据标准化处理公式如下:

$$r_{ij}=\frac{x_{ij}}{\sqrt{\sum_{x=1}^{m}{x_{ij}}^2}} \tag{1}$$

其中, r_{ij}表示第 i 个指标第 j 项数据标准化后的数值, x_{ij}表示第 i 个指标第 j 项数据的初始数值, m 表示每项指标的数据总数.

(2) 通过优序图算法计算各项指标的不同权重. 由于边覆盖数量指标被广泛应用于模糊测试改进评估中, 且和崩溃数量等指标相比具有较高稳定性, 因此将该 4 项指标的权重规定为 2:1:1:1, 计算得到各项指标权重如下:

$$w_i=\begin{cases}0.03977, & 1\leqslant i\leqslant 11\\ 0.01705, & 12\leqslant i\leqslant 44\end{cases} \tag{2}$$

其中, w_1–w_{11} 表示 11 个测试程序的发现边数量指标, w_{12}–w_{44} 则表示剩余指标, 该权重数据由优序图算法计算所得.

根据指标权重及标准化数据计算得到该指标的数据结果, 公式如式 (3):

$$v_{ij}=w_i r_{ij} \tag{3}$$

(3) 计算理想解和负理想解. TOPSIS 算法中的理想解和负理想解分别代表最优及最差的解决方案. 此处最优解决方案和最差解决方案分别指, 存在最理想的模糊测试算法, 其测试数据在各项指标上均取得最高和最低分数, 公式如下:

$$A^+=\left\{v_1^+,v_2^+,\cdots,v_n^+\right\},\quad A^-=\left\{v_1^-,v_2^-,\cdots,v_n^-\right\} \tag{4}$$

其中, A^+和 A^-表示正负理想解, v_i^+和v_i^-表示第 i 项指标权重标准化的最优值和最差值.

(4) 计算每一个候选项, 即模糊测试改进工具的数据与正负理想解之间的距离, 计算公式如下:

$$\begin{cases} D_i^+ = \sqrt{\sum_{j=1}^{m} w_j (v_{ij} - v_j^+)^2} \\ D_i^- = \sqrt{\sum_{j=1}^{m} w_j (v_{ij} - v_j^-)^2} \end{cases} \tag{5}$$

(5) 计算模糊测试改进工具数据距离理想解的贴近程度, 贴近程度越大则该模糊测试改进工具评价效果越好, 计算公式如下:

$$C_i = \frac{D_i^-}{D_i^+ + D_i^-} \tag{6}$$

(6) 根据距离理想解贴近程度, 计算待评估的模糊测试改进工具效果评分并进行排名, 得到整体上各改进工具改进效果评估比较结果.

5 评估实验设置

在建立了模糊测试改进技术评估体系后, 将在该体系基础上对单一模糊测试改进技术进行评估与分析. 本章首先对模糊测试改进技术评估实验的基准模糊测试工具、待评估单一改进技术、测试程序等相关细节进行说明, 然后介绍待评估单一模糊测试改进算法.

5.1 基准模糊测试工具

在单一模糊测试改进技术评估实验中, 选择 AFL 2.52b 作为基准模糊测试工具, 所有的单一改进将均在 AFL 2.52b 基础上进行实现. 如第 3.2 节中所述, AFL 自 2015 年提出以来一直受到广泛关注, 多数模糊测试改进工作选择基于 AFL 改进实现, 几乎所有改进工具都将 AFL 作为评估对比工具. 在 AFL 基础上实现的模糊测试改进工具和算法涵盖了所有的改进类别. 因此, 在本课题的模糊测试改进技术评估中, 选择 AFL 最新稳定版本 AFL 2.52b 作为基准模糊测试工具.

5.2 待评估单一改进技术

为了保证模糊测试改进技术评估的准确性和客观性, 选择的待评估单一模糊测试改进技术需要满足以下条件:

(1) 由于选择 AFL 作为基准模糊测试工具, 算法的原始模糊测试改进工具需基于 AFL 实现;

(2) 算法的原始改进模糊测试工具需开源;

(3) 算法需基于常见的传统模糊测试进行优化, 一些针对性的改进如定向模糊测试改进或内核模糊测试改进不在本次课题进行评估.

基于以上标准, 在第 3.2 节中所述 46 个近年所提出的模糊测试改进工具中, 选择了来自 5 个工具的 8 个单一改进算法, 分别属于 3 种不同改进类别, 改进算法的具体信息如表 1 所示. 其中, *-seed、*-mutation 和*-power 分别表示模糊测试工具*的种子选取、变异和能量分配改进技术.

表 1 待评估模糊测试单一改进技术说明

改进技术	种子选取策略改进	变异策略改进	能量分配策略改进	改进技术描述
AFLFast-seed	√	—	—	优先选择被选择次数少、路径访问次数少的种子
AFLFast-power	—	—	√	选择次数少、路径访问次数少的种子进行更多次数变异
FairFuzz-seed	√	—	—	选择访问了稀有边的种子
FairFuzz-mutation	—	√	—	选择不影响访问稀有边的变异操作和位置
EcoFuzz-seed	√	—	—	在exploration阶段按顺序选择种子, 在exploitation阶段根据SPEM算法选择种子
EcoFuzz-power	—	—	√	在exploration阶段根据AAPS算法分配能量, 在exploitation阶段, 当上次变异发现新路径时分配原有能量, 否则分配双倍能量
MOPT-mutation	—	√	—	根据粒子群算法选择变异操作
BanditAFL-power	—	—	√	根据LSTM算法为种子分配能量

5.3 测试程序

在本次评估中, 共选择了 11 个测试程序, 包括 tcpdump -nr, xmllint, readpng, mutool show, mpg321 --stdout, nm, objdump -d, readelf -a, c++filt, size, catdoc, 具体程序信息如表 2 所示. 所选择的测试程序均在第 3.2 节中所述的 46 个模糊测试改进工作中被广泛应用于效果评估, 且这些程序的输入格式涵盖了近年模糊测试改进评估中的几乎所有格式, 因此将这些程序作为评估的测试程序, 并选择 AFL 所提供的种子文件作为初始种子.

5.4 单一模糊测试改进算法

为了评估所选择的 8 个单一模糊测试改进技术效果, 在选择的基准模糊测试工具 AFL 基础上根据原论文中设计的算法实现相应的 8 个模糊测试工具. 首先

根据原有模糊测试改进工作论文的算法描述, 深入分析相应工具的单一改进算法实现, 按照表 1 所述的单一改进算法划分, 将原有工具的代码实现进行拆分, 基于 AFL 形成新的单一模糊测试改进工具, 通过所建立的模糊测试评估体系进行系统性评估, 从而得到单一改进算法的效果评估.

表 2 模糊测试改进技术评估所使用测试程序

目标程序	参数	程序版本	输入格式
catdoc	—	catdoc-0.93.4	doc
tcpdump	-nr	tcpdump-4.9.0	pcap
xmllint	—	libxml2-2.9.4	xml
readpng	—	libpng-1.9.29	png
mutool	show	mupdf-1.9a-source	pdf
mpg321	--stdout	mpg321-0.3.2.orig	mp3
nm	—	binutils-2.26	executable
objdump	-d	binutils-2.26	executable
readelf	-a	binutils-2.26	elf
c++filt	—	binutils-2.26	stdin
size	—	binutils-2.26	executable

5.5 其他实验条件

本次实验在 Intel(R)Xeon(R) E5-2450 2.10 GHz, 4 核, 64 位 Ubuntu 16.04 系统的计算机上进行测试, 测试时间设定为 72 h, 单个实验重复 10 次取平均值作为实验结果. 参考原有论文, BanditAFL 改进算法在测试中的训练时间选择为 4 h.

6 单一改进技术评估结果与分析

在实现模糊测试改进技术评估体系建立及实验设置后, 我们对各改进类别内的不同单一改进技术进行测试评估与分析. 本节将分别介绍 3 个类别改进算法的评估结果与原因分析.

6.1 种子选取策略改进

在我们选取的单一模糊测试改进算法中, AFLFast-seed、FairFuzz-seed 和 EcoFuzz-seed 属于种子选取策略的改进算法. 实验结果如图 6 所示. 其中, 柱状图数据条上方标注数字与高度分别代表该指标标准化前后的值, 图 7、图 8 数据表示方法相同. 从直观上看, EcoFuzz-seed 相较于其他两个单一改进算法效果较差. 我们基于模糊测试改进技术评估体系对 3 个单一改进工具计算了综合分数, 结果如表 3 所示. 在 3 个算法中, FairFuzz-seed 效果最好, AFLFast-seed 次之, 最后是 EcoFuzz-seed, 与数据观察结果一致.

AFLFast 和 FairFuzz 的种子选取策略核心思想相似, 即优先选择遍历更多低频分支的种子, AFLFast-seed 和 FairFuzz-seed 在评估中效果突出可以反映这种思想的有效性. EcoFuzz 的策略和 AFLFast、FairFuzz 相比具有较大差别. 由 EcoFuzz 的种子选取策略算法可知, EcoFuzz-seed 倾向于选择具有更低自转移频率和队列中更大编号的种子, 自转移频率在其论文中定义为, 种子在变异后仍然覆盖相同路径的概率. 然而, 由一个种子变异生成的样本可能虽然和种子覆盖的原路径不同, 但仍然覆盖了其他种子生成的样本中覆盖的高频路径. 因此, 由此策略选择的种子生成的样本可能包含其他样本覆盖的高频路径, 但仍被优先选择进行变异, 此时选择的种子可能并不能带来更高效的模糊测试效果.

6.2 变异策略改进

在选择的 5 个模糊测试工具中, FairFuzz 和 MOPT 均提出了不同的变异策略优化算法, 期望在对种子进行变异的过程中选择更有效的变异位置和变异操作, 实现更高效的模糊测试. 我们将实现的单一模糊测试改进算法 FairFuzz-mutation 和 MOPT-mutation 在模糊测试改进技术评估体系中进行实验和评估, 实验结果如图 7 所示. 综合上, FairFuzz-mutation 和 MOPT-mutation 的模糊测试效果相似, 在一些测试程序上, 如 mpg321, readpng, size, xmllint, FairFuzz-mutation 效果更好, 而在其他一些程序上, 如 c++filt, mutool, nm, objdump, readelf, MOPT-mutation 具有更好效果. 在评估体系中计算得到的综合评分如表 4 所示. 从评估数据上看, MOPT-mutation 在综合效果上略优于 FairFuzz-mutation.

虽然 FairFuzz-mutation 和 MOPT-mutation 的综合模糊测试效果相似, 但两者的改进策略具有很大不同. FairFuzz-mutation 关注样本中覆盖的低频边, 在确定性变异阶段维护 branch_mask 数组记录低频边访问情况, 在非确定性变异阶段只进行当前变异位置中仍能覆盖到低频边的变异操作. 而 MOPT-mutation 则主要通过粒子群算法选择当前最优的变异操作. 通过数据分析可知, MOPT-mutation 在 11 个测试程序中的 10 个程序具有更高的目标程序执行次数, 一定程度上能够反映 MOPT-mutation 的算法效率更高. 虽然 FairFuzz-mutation 的低频边覆盖策略在提高代码覆盖上效果良好, 但由于其需要一直维护 branch_mask 数组, 带来了相对较大的时间开销, 在综合测试效果上略次于 MOPT-mutation.

6.3 能量分配策略改进

AFLFast、BanditAFL 和 EcoFuzz 均提出了能量分配算法以优化为选定种子分配的变异次数. 测试结

果如图 8 所示, 整体上, BanditAFL-power 的改进效果优于其他两个改进算法, 在 11 个程序中的 8 个程序上 (mpg321, mutool, nm, objdump, readelf, readpng, size, xmllint), BanditAFL 在 4 个指标上的效果均优于或等于其他两个改进算法. 在模糊测试改进技术评估体系上计算综合评估分数如表 5 所示, BanditAFL-power 综合效果最好, 其次为 EcoFuzz-power, 最后是 AFLFast-power.

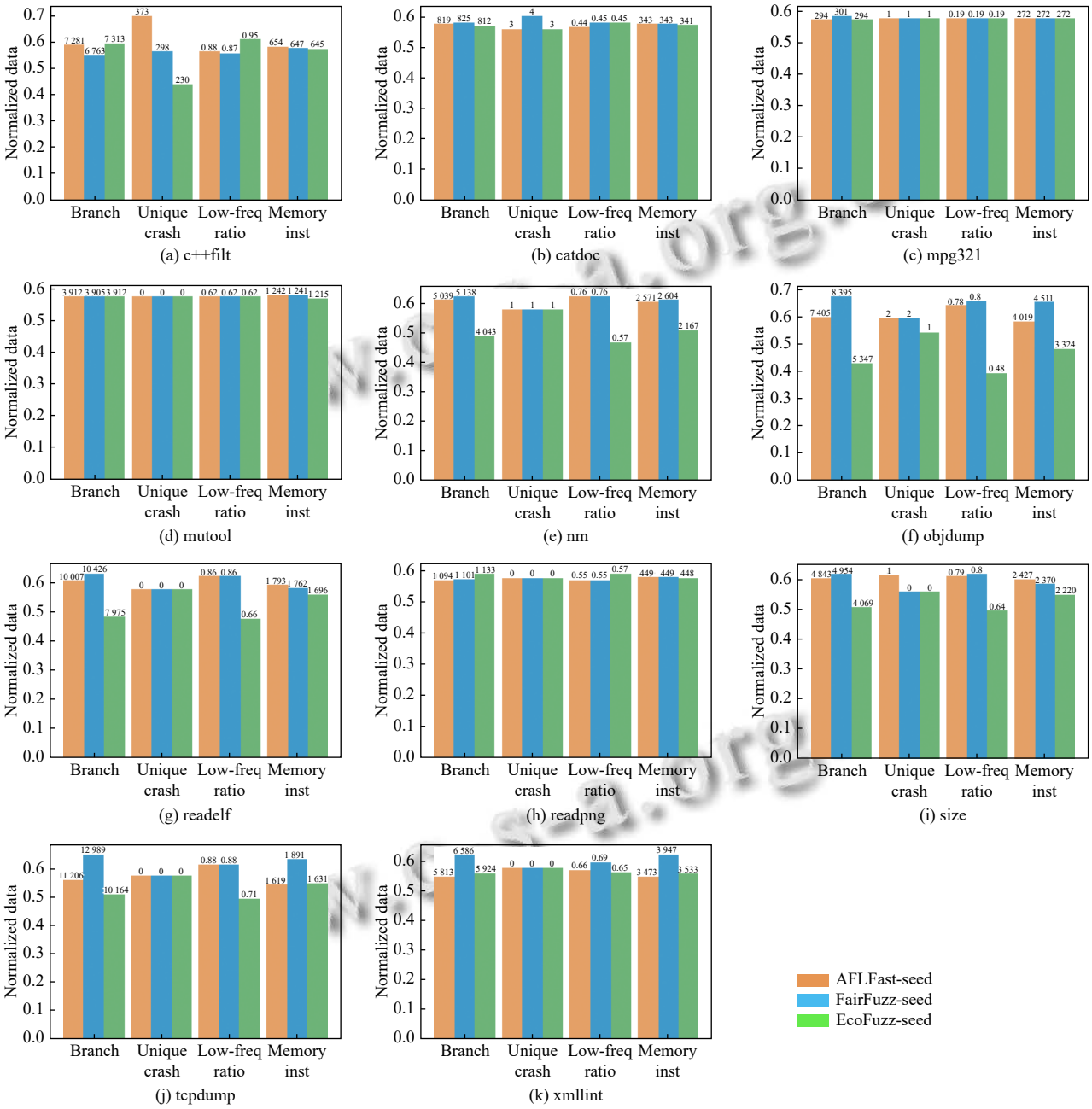

图 6 AFLFast-seed, FairFuzz-seed 与 EcoFuzz-seed 在评估体系 4 个指标测试数据

表 3 3 种模糊测试改进技术的评估分数与排名

模糊测试改进技术	评估分数	排名
FairFuzz-seed	0.737 189	1
AFLFast -seed	0.655 569	2
EcoFuzz-seed	0.344 444	3

BanditAFL-power 采用 LSTM 算法, 根据不同种子的内容优化当前应分配的最优变异次数, 首先通过算法训练得到模型, 在测试过程中通过读取模型数据计算, 得到能量分配结果. 经过数据分析可知, BanditAFL-power

的执行次数较低, 虽然该算法具有较高的时间开销, 但由于根据模型计算得到的能量分配结果更加科学准确, 仍然具有高效的执行结果, 也在一定程度上反映了将时间开销较大的机器学习等算法应用于模糊测试的可行性.

图 7 FairFuzz-mutation 与 MOPT-mutation 在评估体系 4 个指标测试数据

表 4 FairFuzz-mutation 与 MOPT-mutation 评估分数与排名

模糊测试改进技术	评估分数	排名
MOPT-mutation	0.592 726	1
FairFuzz-mutation	0.537 101	2

值得注意的是, 虽然能量分配策略是 AFLFast 原论文中全篇着重介绍的算法, 但其单一改进算法 AFLFast-power 在 3 个能量分配改进算法中效果相对最差, 我们对这一现象进行了进一步分析. 在实验中, 我们应用了 AFLFast 原论文中认为效果最好的 FAST 能量分配策略, 其能量分配计算公式计算如下:

$$p(i)=\min\left(\frac{\alpha(i)}{\beta}\cdot\frac{2^{s(i)}}{f(i)},M\right) \tag{7}$$

其中, $p(i)$表示为当前遍历路径 i 的种子分配的能量, $\alpha(i)$表示原 AFL 计算的能量分配结果, β为常数, 取 $\beta>1$, $s(i)$表示当前种子被选择的次数, $f(i)$表示当前路径被执行的次数, M 为常数, 即能量分配最大值.

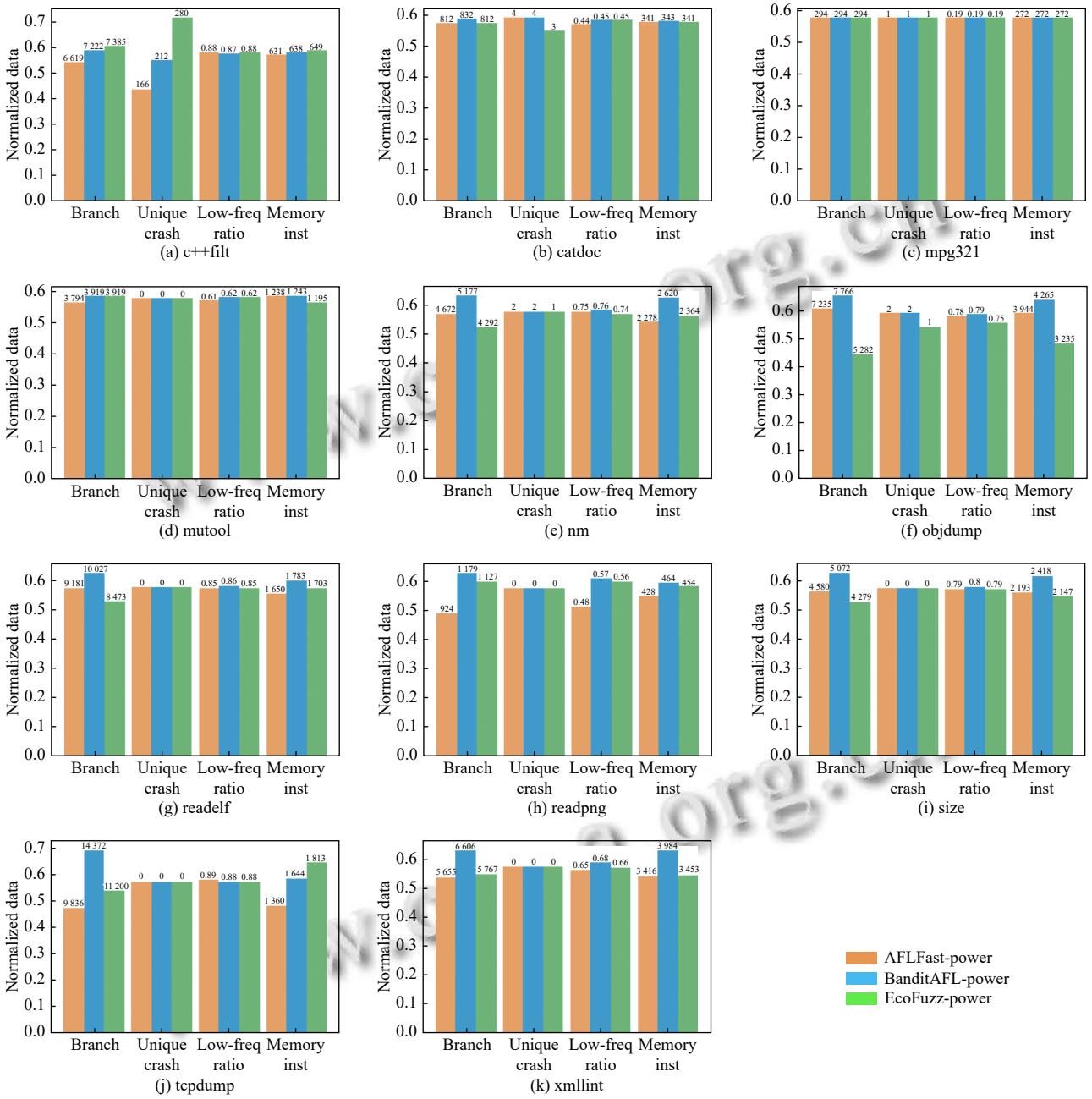

图 8 AFLFast-power, BanditAFL-power 与 EcoFuzz-power 在评估体系 4 个指标测试数据

表 5 3 种模糊测试改进技术的评估分数与排名

模糊测试改进技术	评估分数	排名
BanditAFL-power	0.681 506	1
EcoFuzz-power	0.474 369	2
AFLFast-power	0.427 478	3

AFLFast-power 倾向于为种子选取次数更少、路径执行次数更少的种子分配更多能量. 基于马尔科夫链算法, AFLFast 能够在更短时间内得到高频路径集合, 从而提高模糊测试效率. 然而, 在不结合 AFLFast-seed, 即 AFLFast 的种子选取策略时, 新生成的种子位

于种子队列靠后的位置, 而模糊测试工具仍然按照队列顺序选择种子, 根据式 (7), 一些多次选择的种子仍会被分配到较高的变异次数, 无法在较短时间内快速搜索路径状态, 而浪费了较多时间.

7 相关工作

一些文章对所提出的模糊测试技术进行了总结或评估. Li 等人[1] 对模糊测试中涉及到的不同的技术问题的针对性解决方法进行了整理和总结. Manes 等人[9] 根据模糊测试的不同测试阶段整理了各阶段所提出的模糊测试改进方案. Liang 等人[7] 总结了针对不同应用和不同问题所提出的优化模糊测试工具. Klees 等人[58] 总结并分析了近年来所提出的模糊测试工作中用到的模糊测试评估方法, 并提出了模糊测试效果评估应达到的一些标准. Li 等人[59] 设计实现了一个模糊测试评估系统 UniFuzz, 并对流行的模糊测试工具效果进行了评估. Metzman 等人[10] 基于提出的指标建立了线上模糊测试工具评估平台 FuzzBench. 然而, 当前仍没有工作对模糊测试的单一改进技术进行系统量化的评估与分析.

TOPSIS 算法[57] 为 Hwang 等人于 1981 年首次提出的多指标综合评价方法, 其基本思想为在原始数据标准化处理后, 找到理想最优最劣方案, 计算每个候选项到最优最劣方案的相对接近程度, 从而得到候选项间的综合评价结果. TOPSIS 算法自提出以来, 已在工业与制造业、人力资源管理、商业与市场分析、医学等多个领域应用于多指标综合评估[57], 具有客观性和有效性.

8 结论与展望

本文针对单一模糊测试改进技术评估分析的问题, 基于理论分析与实际经验提出了 4 个评估指标, 建立了客观量化改进技术评估体系. 对近年中模糊测试改进技术进行了整体总结与分析, 从中选择了 5 个先进模糊测试工具中的 8 个单一改进算法, 将这些改进算法分为 3 类, 在所建立的评估体系中进行了实验评估, 将同一改进类别中的不同改进技术进行对比. 我们对实验结果进行了展示与说明, 并结合测试数据与实际算法设计实现进行了深入的原因分析. 实验结果表明:

(1) 在种子选取策略改进算法中, 由于其采用的低频路径、低频边优先改进思想准确有效, AFLFast 和 FairFuzz 具有更好的改进效果;

(2) 在变异策略改进算法中, MOPT 与 FairFuzz 采用不同改进方案, 具有相似改进效果, MOPT 具有更高算法效率和更快执行速度;

(3) 在能量分配策略改进算法中, 采用 LSTM 算法的 BanditAFL 算法改进效果更好, AFLFast 由于算法间存在依赖关系导致单一能量分配算法无法实现应有的效果提升.

在未来的研究工作中, 我们希望继续探究更多模糊测试改进技术相关问题, 如不同类别的改进算法间效果有何差异, 如何组合单一改进算法得到效果最好的模糊测试工具, 同时通过总结分析对未来的模糊测试改进工作提出一些参考意见与建议.

参考文献

1 Li J, Zhao BD, Zhang C. Fuzzing: A survey. Cybersecurity, 2018, 1(1): 6. [doi: 10.1186/s42400-018-0002-y]

2 Xiang LZ, Lin Z. An overview of source code audit. 2015 International Conference on Industrial Informatics-computing Technology, Intelligent Technology, Industrial Information Integration. Wuhan: IEEE, 2015. 26–29.

3 Kim J, Kim T, Im EG. Survey of dynamic taint analysis. 2014 4th IEEE International Conference on Network Infrastructure and Digital Content. Beijing: IEEE, 2014. 269–272. [doi: 10.1109/ICNIDC.2014.7000307]

4 King JC. Symbolic execution and program testing. Communications of the ACM, 1976, 19(7): 385–394. [doi: 10.1145/360248.360252]

5 Zalewski M. American fuzzy lop. http://lcamtuf.coredump.cx/afl/. (2021-06-01).

6 Chen C, Cui BJ, Ma JX, *et al*. A systematic review of fuzzing techniques. Computers & Security, 2018, 75: 118–137.

7 Gan ST, Zhang C, Qin XJ, *et al*. CollAFL: Path sensitive fuzzing. 2018 IEEE Symposium on Security and Privacy (SP). San Francisco: IEEE, 2018. 679–696.

8 Wang MZ, Liang J, Chen YL, *et al*. SAFL: Increasing and accelerating testing coverage with symbolic execution and guided fuzzing. Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings. Gothenburg: IEEE, 2018. 61–64.

9 Manes VJM, Han H, Han C, *et al*. Fuzzing: Art, science, and engineering. arXiv: 1812.00140, 2018.

10 Metzman J, Szekeres L, Simon L, *et al*. FuzzBench: An open fuzzer benchmarking platform and service. Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Athens: ACM, 2021. 1393–1403.

11 Chen P, Chen H. Angora: Efficient fuzzing by principled search. IEEE Symposium on Security and Privacy (SP). San Francisco: IEEE, 2018. 711–725.
12 Böhme M, Pham VT, Roychoudhury A. Coverage-based greybox fuzzing as Markov chain. Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS). Vienna: ACM, 2016. 1032–1043.
13 Lemieux C, Sen K. FairFuzz: A targeted mutation strategy for increasing greybox fuzz testing coverage. 33rd IEEE/ACM International Conference on Automated Software Engineering. Montpellier: IEEE, 2018. 475–485.
14 Yue T, Wang PF, Tang Y, *et al*. EcoFuzz: Adaptive energy-saving greybox fuzzing as a variant of the adversarial multi-armed bandit. Proceedings of the 29th USENIX Conference on Security Symposium. USENIX Association, 2020. 130.
15 Böttinger K, Godefroid P, Singh R. Deep reinforcement fuzzing. 2018 IEEE Security and Privacy Workshops (SPW). San Francisco: IEEE, 2018. 116–122.
16 Lv CY, Ji SL, Zhang C, *et al*. MOPT: Optimized mutation scheduling for fuzzers. 28th USENIX Security Symposium. Santa Clara: USENIX Association, 2019. 1949–1966.
17 Karamcheti S, Mann G, Rosenberg D. Adaptive grey-box fuzz-testing with thompson sampling. Proceedings of the 11th ACM Workshop on Artificial Intelligence and Security. Toronto: ACM. 2018. 37–47.
18 Patil K, Kanade A. Greybox fuzzing as a contextual bandits problem. arXiv: 1806.03806, 2018.
19 Hsu CC, Wu CY, Hsiao HC, *et al*. INSTRIM: Lightweight instrumentation for coverage-guided fuzzing. Workshop on Binary Analysis Research (BAR). San Diego, 2018. 1–7.
20 Xu W, Kashyap S, Min C, *et al*. Designing new operating primitives to improve fuzzing performance. Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. Dallas: ACM, 2017. 2313–2328.
21 Grieco G, Ceresa M, Mista A, *et al*. QuickFuzz testing for fun and profit. Journal of Systems and Software, 2017, 134: 340–354. [doi: 10.1016/j.jss.2017.09.018]
22 Godefroid P, Peleg H, Singh R. Learn&fuzz: Machine learning for input fuzzing. 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE). Urbana: IEEE, 2017. 50–59.
23 Nichols N, Raugas M, Jasper R, *et al*. Faster fuzzing: Reinitialization with deep neural models. arXiv: 1711.02807, 2017.
24 Petsios T, Zhao J, Keromytis AD, *et al*. SlowFuzz: Automated domain-independent detection of algorithmic complexity vulnerabilities. Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. Dallas: ACM, 2017. 2155–2168.
25 Li YK, Chen BH, Chandramohan M, *et al*. Steelix: Program-state based binary fuzzing. Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. Paderborn: ACM, 2017. 627–637.
26 Böhme M, Pham VT, Nguyen MD, *et al*. Directed greybox fuzzing. Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. Dallas: ACM, 2017. 2329–2344.
27 Rajpal M, Blum W, Singh R. Not all bytes are equal: Neural byte sieve for fuzzing. arXiv: 1711.04596, 2017.
28 Drozd W, Wagner MD. FuzzerGym: A competitive framework for fuzzing and learning. arXiv: 1807.07490, 2018.
29 Wüstholz V, Christakis M. Learning inputs in greybox fuzzing. arXiv: 1807.07875, 2018.
30 Greve DA, Gacek A. Trapezoidal generalization over linear constraints. Proceedings of the 15th International Workshop on the ACL2 Theorem Prover and Its Applications. Austin, 2018. 30–46.
31 Lemieux C, Padhye R, Sen K, *et al*. PerfFuzz: Automatically generating pathological inputs. Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis. Daejeon: ACM, 2018. 254–265.
32 Peng H, Shoshitaishvili Y, Payer M. T-Fuzz: Fuzzing by program transformation. 2018 IEEE Symposium on Security and Privacy (SP). San Francisco: IEEE, 2018. 697–710.
33 Kargén U, Shahmehri N. Speeding up bug finding using focused fuzzing. Proceedings of the 13th International Conference on Availability, Reliability and Security. Hamburg: ACM, 2018. 7.
34 Chen HX, Xue YX, Li YK, *et al*. Hawkeye: Towards a desired directed grey-box fuzzer. Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. Toronto: ACM, 2018. 2095–2108.
35 Chen HX, Li YK, Chen BH, *et al*. Fot: A versatile, configurable, extensible fuzzing framework. Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Lake Buena Vista: ACM, 2018. 867–870.
36 Jain V, Rawat S, Giuffrida C, *et al*. TIFF: Using input type inference to improve fuzzing. Proceedings of the 34th Annual Computer Security Applications Conference. San Juan: ACM, 2018. 505–517.

37 Zhang B, Ye JX, Bi X, *et al*. Ffuzz: Towards full system high coverage fuzz testing on binary executables. PLoS One, 2018, 13(5): e0196733. [doi: 10.1371/journal.pone.0196733]

38 Liang J, Jiang Y, Chen YL, *et al*. PAFL: Extend fuzzing optimizations of single mode to industrial parallel mode. Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Singapore: ACM, 2018. 809–814.

39 You W, Liu XW, Ma SQ, *et al*. SLF: Fuzzing without valid seed inputs. 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). Montreal: IEEE, 2019. 712–723.

40 Fu Y, Tong SM, Guo XY, *et al*. Improving the effectiveness of grey-box fuzzing by extracting program information. 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom). Guangzhou: IEEE, 2020. 434–441.

41 Aschermann C, Schumilo S, Blazytko T, *et al*. REDQUEEN: Fuzzing with input-to-state correspondence. Network and Distributed Systems Security (NDSS) Symposium 2019. San Diego: The Internet Society, 2019. 1–15.

42 You W, Wang XQ, Ma SQ, *et al*. ProFuzzer: On-the-fly input type probing for better zero-day vulnerability discovery. 2019 IEEE Symposium on Security and Privacy (SP). San Francisco: IEEE, 2019. 769–786.

43 She DD, Pei KX, Epstein D, *et al*. NEUZZ: Efficient fuzzing with neural program smoothing. 2019 IEEE Symposium on Security and Privacy (SP). San Francisco: IEEE, 2019. 803–817.

44 Zhao L, Duan Y, Yin H, *et al*. Send hardest problems my way: Probabilistic path prioritization for hybrid fuzzing. Network and Distributed Systems Security (NDSS) Symposium 2019. San Diego: The Internet Society, 2019. 1–15.

45 Nagy S, Hicks M. Full-speed fuzzing: Reducing fuzzing overhead through coverage-guided tracing. 2019 IEEE Symposium on Security and Privacy (SP). San Francisco: IEEE, 2019. 787–802.

46 Chen YL, Jiang Y, Ma FC, *et al*. EnFuzz: Ensemble fuzzing with seed synchronization among diverse fuzzers. 28th USENIX Security Symposium. Santa Clara: USENIX Association, 2019. 1967–1983.

47 Wang YH, Jia XK, Liu YW, *et al*. Not all coverage measurements are equal: Fuzzing by coverage accounting for input prioritization. Network and Distributed Systems Security (NDSS) Symposium 2020. San Diego: The Internet Society, 2020. 1–17.

48 Gan ST, Zhang C, Chen P, *et al*. GREYONE: Data flow sensitive fuzzing. 29th USENIX Security Symposium. USENIX Association, 2020. 2577–2594.

49 Österlund S, Razavi K, Bos H, *et al*. ParmeSan: Sanitizer-guided greybox fuzzing. 29th USENIX Security Symposium. USENIX Association, 2020. 2289–2306.

50 Chen YH, Ahmadi M, Farkhani RM, *et al*. MEUZZ: Smart seed scheduling for hybrid fuzzing. 23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID). San Sebastian: USENIX Association, 2020. 77–92.

51 Wang JH, Song CY, Yin H. Reinforcement learning-based hierarchical seed scheduling for greybox fuzzing. 28th Annual Network and Distributed System Security Symposium. The Internet Society, 2021. 1–17.

52 Mi XY, Rawat S, Giuffrida C, *et al*. LeanSym: Efficient hybrid fuzzing through conservative constraint debloating. 24th International Symposium on Research in Attacks, Intrusions and Defenses. San Sebastian: ACM, 2021. 62–77.

53 Huang HQ, Yao PS, Wu RX, *et al*. Pangolin: Incremental hybrid fuzzing with polyhedral path abstraction. 2020 IEEE Symposium on Security and Privacy (SP). San Francisco: IEEE, 2020. 1613–1627.

54 Aschermann C, Schumilo S, Abbasi A, *et al*. Ijon: Exploring deep state spaces via fuzzing. 2020 IEEE Symposium on Security and Privacy (SP). San Francisco: IEEE, 2020. 1597–1612.

55 Fu Y, Shi DH, Zhang Y, *et al*. Improved fuzz testing approach based on coverage frequency. Computer Systems & Applications, 2019, 28(1): 17–24.

56 Wikipedia. Precedence graph. https://en.wikipedia.org/wiki/Precedence_graph. (2021-05-21) [2021-06-01].

57 Behzadian M, Otaghsara SK, Yazdani M, *et al*. A state-of the-art survey of TOPSIS applications. Expert Systems with Applications, 2012, 39(17): 13051–13069. [doi: 10.1016/j.eswa.2012.05.056]

58 Klees G, Ruef A, Cooper B, *et al*. Evaluating fuzz testing. Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. Toronto: ACM, 2018. 2123–2138.

59 Li YW, Ji SL, Chen Y, *et al*. Unifuzz: A holistic and pragmatic metrics-driven platform for evaluating fuzzers. 30th USENIX Security Symposium (USENIX Security 21). USENIX Association. 2021. 2777–2794.

(校对责编: 牛欣悦)