



# 计算机科学

COMPUTER SCIENCE

## 基于强化学习的分布式Android应用自动化测试方法

宋日荣, 陈钦文, 陈星

引用本文

宋日荣, 陈钦文, 陈星. 基于强化学习的分布式Android应用自动化测试方法[J]. 计算机科学, 2025, 52(12): 40-47.

SONG Rirong, CHEN Qinwen, CHEN Xing. [Distributed Automated Testing for Android Applications Based on Reinforcement Learning](#) [J]. Computer Science, 2025, 52(12): 40-47.

---

## 相似文章推荐 (请使用火狐或 IE 浏览器查看文章)

Similar articles recommended (Please use Firefox or IE to view the article)

### [改进深度强化学习的多智能体联合导航策略研究](#)

Research on Multi-agent Joint Navigation Strategy Based on Improved Deep Reinforcement Learning  
计算机科学, 2025, 52(11A): 250200095-7. <https://doi.org/10.11896/jsjcx.250200095>

### [利用融合2-opt的强化学习算法求解TSP问题](#)

Hybrid Reinforcement Learning Algorithm Combined with 2-opt for Solving Traveling Salesman Problem  
计算机科学, 2025, 52(11A): 250200121-8. <https://doi.org/10.11896/jsjcx.250200121>

### [基于联盟区块链的数据可信共享方案](#)

Data Trusted Sharing Scheme Based on Consortium Blockchain  
计算机科学, 2025, 52(11): 398-407. <https://doi.org/10.11896/jsjcx.241000169>

### [基于汤普森采样的自适应安卓程序测试方法](#)

Adaptive Android Program Test Method Based on Thompson Sampling  
计算机科学, 2025, 52(11): 330-338. <https://doi.org/10.11896/jsjcx.240900150>

### [基于卷积双延迟深度确定性策略梯度的卫星网络多路径路由算法](#)

Multipath Routing Algorithm for Satellite Networks Based on Convolutional Twin Delay Deep Deterministic Policy Gradient  
计算机科学, 2025, 52(11): 280-288. <https://doi.org/10.11896/jsjcx.240800161>

# 基于强化学习的分布式 Android 应用自动化测试方法

宋日荣 陈钦文 陈星

福州大学计算机与大数据学院 福州 350116

大数据智能教育部工程研究中心 福州 350116

福建省网络计算与智能信息处理重点实验室(福州大学) 福州 350116

(songriron@foxmail.com)

**摘要** Android 应用程序已经深深融入人们日常生活的方方面面,确保这些应用的正确性是一个极具挑战性的任务。传统测试方法主要依赖于手工操作,自动化测试技术尽管有所发展但仍有待改进。Android 应用程序不断迭代以完善性能和功能需求,导致了应用程序复杂性的增加和状态组合的爆炸性增长。测试 Android 应用的核心在于探索复杂的用户交互下的深层故障,但是这些故障的搜索空间是巨大的,需要花费大量的时间来进行测试。近年来,研究人员开始使用强化学习来测试 Android 应用,利用智能体与 Android 应用交互过程中获得的奖励来调整探索策略。然而,现有工作仅利用单台设备进行测试,测试效率十分有限。为了应对上述挑战,提出了一种基于强化学习的分布式 Android 应用自动化测试框架 DistributedAndroidExplore(DAE),利用多个智能体同时对应用程序进行基于强化学习的测试,并定期迭代地聚合每个智能体累积的学习经验,以此提高测试效率。在 10 个真实世界的 Android 应用程序上对 DAE 进行了评估,结果表明,在大多数情况下,DAE 的故障检测率、代码覆盖率均优于所对比的基准算法。同时测试效率明显优于其他方法,性能提高了 16.5%~34.3%。

**关键词:** 分布式系统; Android 应用程序; 强化学习; Q-learning; 自动化测试

**中图分类号** TP311

## Distributed Automated Testing for Android Applications Based on Reinforcement Learning

SONG Rirong, CHEN Qinwen and CHEN Xing

College of Computer and Data Science, Fuzhou University, Fuzhou 350116, China

Engineering Research Center of Big Data Intelligence, Ministry of Education, Fuzhou 350116, China

Fujian Key Laboratory of Network Computing and Intelligent Information Processing(Fuzhou University), Fuzhou 350116, China

**Abstract** Android applications have become deeply integrated into various aspects of people's daily lives. However, ensuring the correctness of these applications remains a highly challenging task. Traditional testing methods primarily rely on manual operations, while automated testing technologies, despite advancements, still require improvements. The continuous iteration of Android applications to enhance performance and meet functional requirements has led to increased application complexity and an explosive growth in state combinations. Testing the core aspects of Android applications involves exploring deep-seated failures under complex user interactions, but the search space for these failures is vast, necessitating substantial time investment for thorough testing. In recent years, researchers have begun employing reinforcement learning to test Android applications by adjusting exploration strategies based on rewards obtained during the interaction between agents and the Android applications. However, existing work has been limited to testing with a single device, significantly constraining testing efficiency. To address these challenges, this paper proposes a distributed automated testing framework for Android applications based on reinforcement learning, named DistributedAndroidExplore(DAE). DAE utilizes multiple agents to concurrently conduct reinforcement learning-based testing on applications and periodically iterates to aggregate the cumulative learning experiences of each agent, thereby enhancing testing efficiency. DAE is evaluated on 10 real-world Android applications. The results indicate that, in most cases, DAE surpasses the compared benchmark algorithms in terms of fault detection rate and code coverage. Furthermore, DAE demonstrates notably higher

到稿日期:2024-11-08 返修日期:2025-01-23

基金项目:国家自然科学基金(62072108);福建省促进海洋与渔业产业高质量发展专项资金(FJHYF-ZH-2023-02);福建省技术创新重点攻关及产业化项目(2024XQ004)

This work was supported by the National Natural Science Foundation of China(62072108), Special Funds for Promoting High-quality Development of Marine and Fishery Industries in Fujian Province (FJHYF-ZH-2023-02) and Fujian Key Technological Innovation and Industrialization Projects (2024XQ004).

通信作者:陈星(chenxing@fzu.edu.cn)

testing efficiency, with performance improvements ranging from 16.5% to 34.3%.

**Keywords** Distributed system, Android applications, Reinforcement learning, Q-learning, Automated testing

## 1 引言

随着移动设备的普及,移动应用程序融入了用户的日常生活。据报道<sup>[1]</sup>称,用户平均每日移动应用程序的使用时长超过 2 小时,这一数据凸显了确保应用程序正确性的重要性。在实践中,程序员通常通过测试来提高应用程序的正确性。尽管通过测试发现了许多严重的故障,但测试 Android 应用程序仍然具有挑战性。例如,如果一个应用程序包含复杂的业务功能、多个界面和可执行事件,那么生成测试用例就需要枚举大量可能的事件和转换的组合空间。

为了生成高质量的 Android 应用测试用例,研究人员提出了多种方法,这些方法大致可分为随机测试、基于模型的测试和系统测试。尽管这些方法成功发现了许多故障,但它们仍存在内在的局限性。随机测试<sup>[2-4]</sup>生成的是伪随机事件,可能会生成无效事件,并且可能无法触发复杂的事件。基于模型的测试<sup>[5-8]</sup>采用动态或静态策略构建 Android 应用的模型,并利用该模型指导测试用例的生成。然而,由于 Android 应用可能具有复杂的状态和行为,构建完整的应用模型非常困难。如果模型不完整,则生成的测试用例的质量可能会受到影响。系统测试<sup>[9-11]</sup>使用符号执行等复杂技术来指导测试用例的生成。这些方法的可扩展性较差,在实际开发中的有效性常常受到质疑。

近年来,研究人员开始使用强化学习<sup>[12-16]</sup>来测试 Android 应用程序。Android 测试的目标是探索更多的状态并发现应用程序中隐藏的故障。为了达到这一目标,强化学习利用代理与真实环境(即 Android 应用程序)交互过程中获得的奖励来调整探索策略。这些故障的搜索空间巨大,需要花费大量的时间来进行探索。因此,测试效率对在给定的测试时间预算内实现有效测试具有至关重要的意义。然而,现有工作仅利用单台设备进行测试,测试效率十分有限。

为了应对上述挑战,受现有分布式技术<sup>[8,17]</sup>启发,本文提出了一个名为 DistributedAndroidExplore (DAE) 的分布式 Android 测试框架。DAE 利用多个智能体同时对应用程序进行基于强化学习的测试并构建了分布式测试框架,来自适应地探索 Android 程序并生成高质量测试用例。特别地,DAE 在不断完善探索策略的同时进行了即时测试。这与传统的人工智能方法<sup>[18-19]</sup>不同,后者仅限于在训练后进行应用。一方面,为了实现高覆盖的测试,在每个智能体上实现了基于 Android 界面结构的状态抽象,并采用 Android 应用状态相似度计算方法<sup>[20]</sup>设计了一个好奇心驱动的奖励函数,训练强化学习策略去探索更多应用程序行为。另一方面,为了实现高效率的测试,DAE 通过构建分布式测试框架,利用多个智能体同时对应用程序进行基于强化学习的测试,并定期迭代地聚合每个智能体累积的学习经验,以此提升测试效率。

本文的主要贡献如下:

1) 提出了一个基于强化学习的分布式 Android 测试框架 DAE,利用多个智能体同时对应用程序进行基于强化学习的

测试。对于每个智能体,利用 Android 应用状态相似度计算方法设计了一个更精细、更有效的奖励函数,并以此引导探索走向更多新场景。同时定期迭代地聚合每个智能体累积的学习经验,以此引导探索更快走向更多新场景。

2) 在 10 个开源 Android 应用程序上全面评估了 DAE。结果表明,在大多数情况下,DAE 的故障检测率、代码覆盖率均优于所对比的基准算法。同时测试效率明显优于其他方法,性能提高了 16.5%~34.3%。

## 2 Android 应用程序和强化学习

Android 应用程序通常需要执行一系列操作(例如点击或输入文本)来进行交互,而执行的操作将改变 Android 应用程序的状态(例如 Activity 或 GUI)。此外,与 Web 应用不同,Android 应用中的用户交互可能包含许多移动设备特有的系统级事件(例如音量增大或减小等)。这一过程可以形式化为马尔可夫决策(MDP)。MDP 可以定义为一个 4 元组  $\langle S, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$ ,其中  $S$  是状态集,  $\mathcal{A}$  是动作集,  $\mathcal{P}$  是概率转换函数,  $\mathcal{R}$  是奖励函数。在一个离散时间步骤  $t$ ,智能体观察 Android 应用程序状态  $s_t \in S$ ,然后根据策略函数  $\pi(\cdot)$  选择并执行一个动作  $a_t \in \mathcal{A}$ 。在此之后,Android 应用程序转换到一个新状态  $s_{t+1} = \mathcal{P}(s_t, a_t)$ ,同时智能体得到一个即时奖励  $r_t = \mathcal{R}(s_t, a_t)$ 。

由此,智能体根据策略函数  $\pi(\cdot)$  选择动作  $a_t \in \mathcal{A}$ ,并与 Android 应用程序交互得到以下轨迹。

$$traj = (s_0, a_0, r_0, \dots, s_t, a_t, r_t, \dots)$$

其中,下标表示不同的离散时间步骤。每个轨迹都有一个累积奖励,通常被定义为  $R_t = \sum_{t' > 0} \gamma^{t'-t} r_{t'}$ ,  $\gamma \in [0, 1]$  表示未来奖励的折扣因子。

一般来说,Android 应用测试的目标是探索更多的 Android 应用状态,发现应用中隐藏的故障。强化学习能够在与 Android 应用程序的交互中获取奖励。当一个动作导致发现新的应用状态时,该动作会获得较高的好奇心奖励。这能够增加探索到新状态的概率,使得更多的不同状态被触及,进而有助于发现更多隐藏的故障。

## 3 分布式 Android 应用自动化测试

### 3.1 方法概述

本文提出了一个基于强化学习的分布式 Android 测试框架 DistributedAndroidExplore (DAE),旨在利用多个智能体同时对应用程序进行基于强化学习的测试并构建分布式测试框架,以自适应地探索 Android 程序并生成高质量测试用例。

该方法主要包含各个智能体对被测应用的测试和聚合两个步骤。在智能体测试步骤中,DAE 分布式执行一组测试智能体,这些智能体彼此独立执行,并定期聚合它们的学习经验。将智能体独立测试的时间段称为“回合”,在每个回合结束时,所有智能体会将 Q 表、动作执行次数发送给控制器。在聚合步骤中,聚合器从控制器接收各智能体的数据,据此计

算出一个代表累计学习经验的 Q 表,并将聚合后的 Q 表传回,用于下一回合的测试。

### 3.2 智能体测试

图 1 展示了智能体测试流程的概览,整个过程可以分为两个主要部分:1)左侧部分展示的是预处理模块,该模块负责

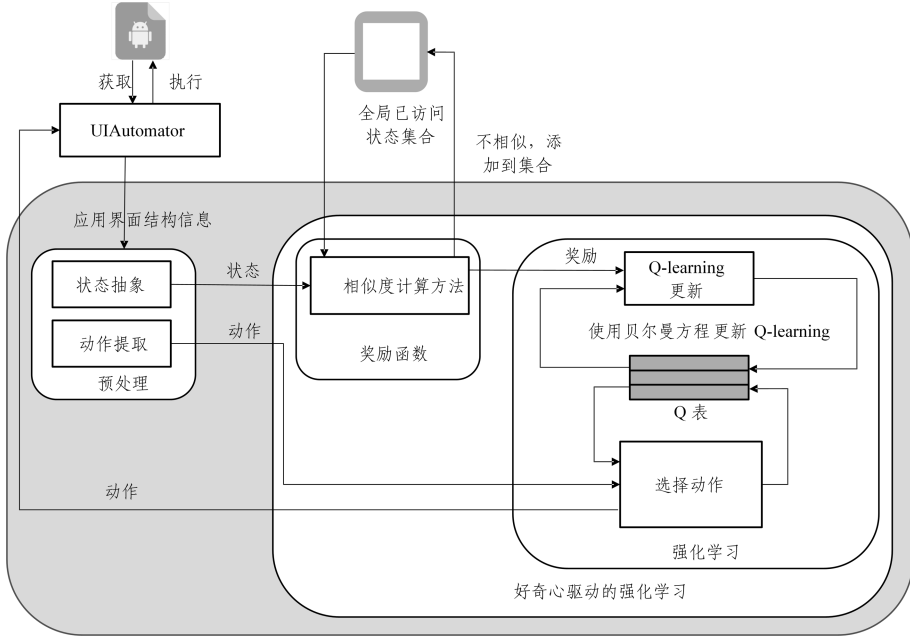


图 1 智能体测试步骤概览

Fig. 1 Overview of the steps of the agent testing

算法 1 详细描述了智能体的测试步骤。在给定的测试期间内,智能体持续探索被测应用(第 1 行)。为了避免陷入无法继续的状态转移,设置了动作序列的最大步数(第 5 行),一旦达到上限,智能体会重启应用以开始新一轮的测试。所有的智能体共同维护一个全局已访问状态集  $M$ ,用于记录已访问过的状态(第 4 行),以此指引强化学习去探索未访问过的状态,即好奇的状态。预处理模块从被测应用中抽象出初始 Android 应用状态  $s_t$  和动作集合  $A_t$ (第 3 行)。每一轮的测试开始于初始状态,根据策略选择并执行动作,被测应用跳转到下一状态,同时更新访问计数表(第 6-9 行)。若检测到故障状态,则将导致故障的动作序列加入故障用例集合(第 10-13 行)。若没有检测到故障,则继续抽象下一 Android 应用状态  $s_{t+1}$  和动作集合  $A_{t+1}$ (第 14 行)。当被测应用转换至新状态  $s_{t+1}$  后,根据 Android 应用状态相似度计算方法,计算当前状态与全局已访问状态集的相似度并计算奖励,同时更新状态集合(第 15-20 行)。然后,根据此状态转换训练强化学习策略  $\pi$ (第 22 行)。当智能体测试的回合结束时,智能体准备与其他智能体聚合其学习经验(第 24-25 行)。最后,当 Android 应用测试结束时,DAE 使用 Jacoco 工具来统计此次测试中的代码覆盖率(第 29 行)。

#### 算法 1 智能体测试

输入:被测 Android 应用 AUT,总测试时长  $T_{total}$ ,每回合测试时长

$T_{round}$ ,动作序列最大步数  $L$ ,相似度阈值  $threshold$

输出:发现 Android 应用故障的测试用例集合  $F$ ,代码覆盖率  $P$

初始化:强化学习策略  $\pi$ ,全局已访问状态集合  $M = \emptyset$ ,发现 Android 应用故障的测试用例集合  $F = \emptyset$ ,测试用例轨迹  $traj = \emptyset$ ,当前

基于 Android 应用界面结构实现状态抽象和动作分析(详细介绍见 3.3 节);2)右侧部分的好奇心驱动的强化学习模块,所有智能体共同维护一个全局已访问状态集,将 Android 应用状态相似度计算方法融入奖励函数的设计中,并以此指导 Q-learning 探索策略的优化(详细介绍见 3.4 节)。

执行的时间步骤  $t=0$ ,访问计数表  $N=0$

```

1. while  $\neg$  timeout( $T_{total}$ ) do
2.   reset(AUT)
3.    $s_t, A_t \leftarrow$  preprocessing(AUT, M)
4.    $M \leftarrow M \cup s_t$ 
5.   for each  $t \in [1, L]$  do
6.      $a_t \leftarrow$  getAction( $\pi, s_t$ )
7.      $traj \leftarrow$  traj.append( $s_t, a_t$ )
8.      $failed \leftarrow$  execute(AUT,  $a_t$ )
9.      $N(s_t, a_t) = N(s_t, a_t) + 1$ 
10.    if failed then
11.       $F \leftarrow F \cup traj$ 
12.      break
13.    end if
14.     $s_{t+1}, A_{t+1} \leftarrow$  preprocessing(AUT, M)
15.    for  $s$  in  $M$  do
16.      similarity  $\leftarrow$  max(sim( $s_{t+1}, s$ ))
17.    end for
18.    if similarity  $<$  threshold then
19.       $M \leftarrow M \cup s_{t+1}$ 
20.    end if
21.     $r_t \leftarrow$  R( $s_t, a_t$ )
22.    根据( $s_t, a_t, s_{t+1}, r_t$ )训练强化学习策略  $\pi$ 
23.     $s_t = s_{t+1}$ 
24.    if  $\neg$  timeout( $T_{round}$ ) then
25.      将信息传给控制器并等待反馈
26.    end if
27.  end for

```

28. end while

29. 测试结束,根据被测应用程序 AUT 获取代码覆盖率 P

### 3.3 预处理

若要通过强化学习进行策略学习,则必须定义状态表示。虽然直接使用 Android 应用程序的屏幕图像(即截图)<sup>[21]</sup>作为状态表示是一种直观的方式,但考虑到 Android 应用的动态特性,这种方法可能导致状态空间的爆炸性增长。例如,用户的不同交互可能会导致屏幕图像的变化,即便这些变化仅限于视觉层面而不涉及业务逻辑。同时还观察到,具有相同业务逻辑的页面通常具有相似的界面结构。因此,根据 Android 应用程序的界面结构来抽象化其状态。具体而言,利用自动化框架(UIAutomator)分析当前应用界面的结构,提取 Activity 和各类视图组件的信息,以此抽象出 Android 应用状态  $s_t = (act, e_1, e_2, \dots, e_n)$ , 其中  $act$  表示当前 Android 应用界面的 Activity,  $e_i = \{path_i, classname_i, x_i, y_i, width_i, height_i\}$  表示界面中的视图组件,它们以树型结构排列,其中  $path_i$  表示元素  $e_i$  在 Android 界面结构中的所在路径,  $classname_i$  表示  $e_i$  的类名,  $x_i, y_i, width_i, height_i$  分别表示  $e_i$  在界面中的横坐标、纵坐标、宽度和高度。

算法 2 详细描述了预处理模块的工作流程。预处理模块以 AUT 为输入,输出当前状态  $s_t$  和动作集合  $A_t$ 。在测试过程中,所有的智能体共同维护一个全局已访问状态集  $M$ ,用于记录已访问过的状态。首先,该模块利用自动化框架(UIAutomator)提取应用界面结构信息,并根据这些信息将其映射成一个应用状态  $s_t$ (第 1—2 行)。其次,该模块遍历并分析每个视图控件的属性(例如可点击、可滚动等),据此推断并生成可执行动作集合(第 3 行)。此外,还将移动设备上的系统级事件(例如屏幕旋转、音量控制等)纳入动作集合,以揭示更多潜在的应用故障(第 4 行)。最后,为了避免状态空间的无限增长,若当前状态  $s_t$  与已访问状态  $s$  的相似度大于所设定阈值,则返回状态  $s$ , 否则返回新状态  $s_t$ (第 5—11 行)。

#### 算法 2 预处理

输入: 被测的 Android 应用程序 AUT, 全局已访问状态集合  $M$

输出: 当前 Android 应用状态  $s_t$ , 当前应用状态下的有效动作集合  $A_t$

1.  $activity, pageInfo \leftarrow dumpAndAnalysis(AUT)$
2. 使用  $\langle activity, pageInfo \rangle$  创建 Android 应用状态  $s_t$
3.  $A_t \leftarrow analysisActionWithUIAutomator(AUT)$
4. 添加移动设备的系统事件到动作集合  $A_t$
5. for  $s$  in  $M$  do
6.    $similarity = sim(s_t, s)$
7.   if  $similarity > threshold$  then
8.     return  $s, A_t$
9.   end if
10. end for
11. return  $s_t, A_t$

### 3.4 好奇心驱动的强化学习

在典型的强化学习任务中,通常存在一个明确且可量化的优化目标,这为奖励函数的设计提供了清晰的依据。然而,在 Android 应用测试领域,设计有效的奖励函数并不容易。这是因为 Android 应用测试的核心目标是尽可能探索应用程序中的不同行为,这一目标较为模糊,难以直接量化。为了应

对这一挑战,注意到具有不同功能的 Android 应用页面往往具有不同的界面结构,因此提出了一种有效的奖励函数设计,旨在鼓励智能体探索新的应用状态。

文献[20]提出了一种 Android 应用状态相似度方法,能够计算两个 Android 状态的相似度  $sim(s_1, s_2)$ 。基于该方法,本文设计了一种好奇心驱动的奖励函数,通过引入适应性探索策略来激励对新状态的发现。具体而言,在测试期间,所有智能体共同维护一个全局已访问状态集合  $M$  来记录所有已访问过的状态。当到达状态  $s_t$  时,若状态  $s_t$  的 Activity 与  $M$  中所有已访问状态的 Activity 都不同,或  $s_t$  是故障状态,则认为发现了全新的状态,并给予 100 的正向奖励;否则,将当前状态  $s_t$  与  $M$  中具有相同 Activity 的所有状态进行相似度比较,根据最大相似度值给予奖励:相似度越高,奖励越低;若相似度超过预设阈值,则认为该状态已被访问过,其探索价值较低,此时给予 -100 的负向奖励以避免重复探索。好奇心驱动的奖励函数如式(1)所示:

$$R(s_t) = \begin{cases} 100, & g(s_t) \notin g(M) \text{ or } crash \\ 100 \cdot (1 - h(s_t)), & g(s_t) \in g(M) \text{ and } h(s_t) \leq 0.9 \\ -100, & g(s_t) \in g(M) \text{ and } h(s_t) > 0.9 \end{cases} \quad (1)$$

其中,  $s_t$  表示当前状态,  $M$  表示全局已访问状态集合,  $g(s)$  函数表示状态  $s$  的 Activity,  $crash$  表示应用故障状态,  $h(s) = \max_{\substack{t \in M \\ g(t) = g(s)}} sim(s, t)$  是状态  $s$  与集合  $M$  中已访问状态的最大相似度。

本文中,DAE 采用了一种无模型的强化学习方法——Q-learning,以优化基于好奇心驱动奖励的策略。Q-learning 通过维护一个 Q 函数来评估状态-动作对的价值,该 Q 函数为每一对状态和动作分配一个 Q 值,表示从当前状态执行该动作后预期能获得的累积奖励。由于 Android 应用测试中的状态和动作均为离散型,使用 Q 表来表示 Q 函数。在每一个离散的时间步骤  $t$  内,DAE 依据环境反馈的状态转换信息  $(s_t, a_t, s_{t+1})$  以及由好奇心驱动的奖励函数计算出的奖励值  $r_t$  更新 Q 表(见式(2)),从而动态调整强化学习的探索策略  $\pi$ 。

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \quad (2)$$

Q-learning 依据 Q 函数遵循  $\epsilon$ -greedy 策略来选取在状态  $s_{t+1}$  的下一步有效动作  $a_{t+1}$ , 动作的选择策略如式(3)所示:

$$getAction(s) = \begin{cases} \arg \max_a Q(s, a), & 1 - \epsilon \\ \text{random UI event}, & \frac{1}{2} \epsilon \\ \text{random system event}, & \frac{1}{2} \epsilon \end{cases} \quad (3)$$

其中,以  $1 - \epsilon$  的概率选取具有最大 Q 值的动作,分别以  $\frac{1}{2} \epsilon$  的概率随机选取 UI 事件或系统事件作为下一步动作。

由此,在强化学习指导下的 Android 应用自动化测试中,当发现新的状态时,该动作会被赋予较高的好奇心奖励,从而激励智能体优先探索尚未访问的状态,增加发现新状态的概率。DAE 进一步将 Android 应用状态相似度融入奖励函数设计,使得与已访问状态相似度越高的状态,其探索价值越

低,相应的好奇心奖励也越小。这能够鼓励智能体专注于探索那些更具独特性和潜在价值的状态。此外,当两个智能体先后首次访问同一状态时,首先到达的智能体将获得 100 的正向奖励,该状态随即被加入全局已访问状态集合;而后到达的智能体由于重复访问已知状态,将获得 -100 的负向奖励。这样的设置有效地驱使各个智能体分散探索,避免冗余搜索,提高了整个系统的探索效率和扩大了覆盖范围。

### 3.5 聚合

聚合步骤的概览如图 2 所示。在每一回合结束时,所有智能体会将 Q 表、访问计数表发送给控制器。当所有的智能体都将数据传至控制器后,聚合器从控制器接收各智能体的数据,据此计算出一个代表累计学习经验的 Q 表,并将聚合后的 Q 表传回,用于下一回合的测试。算法 3 给出了聚合步骤的详细过程。聚合器将每个智能体的 Q 表和访问计数表作为输入,并输出聚合后的 Q 表。聚合器按照“少数服从多数”的原则,关于多个智能体对同一动作的学习经验,它更信任该动作执行次数更多的智能体(第 1—6 行)。

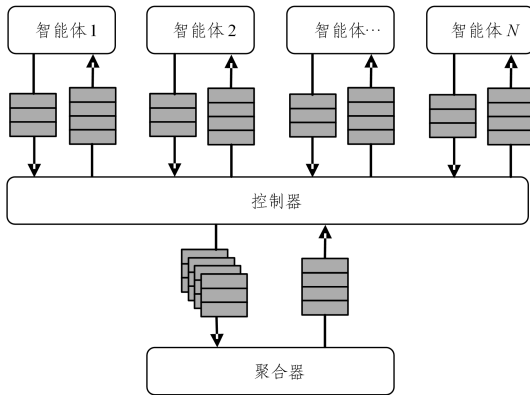


图 2 聚合步骤概览图

Fig. 2 Overview of the steps of aggregation

#### 算法 3 聚合

输入: 每个智能体的 Q 表  $Q_{1,2,\dots,n}$ , 每个智能体的访问计数表  $N_{1,2,\dots,n}$

输出: 聚合后的 Q 表

```

1. for  $(s_t, a_t)$  in  $Q_{1,2,\dots,n}$  do
2.   if  $N_i(s_t, a_t) \geq N_{1,2,\dots,n}(s_t, a_t)$  then
3.      $Q(s_t, a_t) = Q_i(s_t, a_t)$ 
4.   end if
5. end for
6. return Q

```

## 4 实验分析

### 4.1 实验设置

本文基于 Python 3.8, UIAutomator2<sup>[22]</sup> 和 Xposed 实现了 DAE。为了验证本文方法的有效性,构建了一个由 10 个真实世界 Android 应用程序组成的研究基准,并在此基础上将本文方法与现有算法进行了对比分析。大部分应用选自 Q-testing<sup>[12]</sup>。由于有些应用已经过时或无法编译,本文并未选用所有应用。从 GitHub 中收集了一些正在积极维护中的应用程序,以补充形成新的研究基准。这 10 个真实的 Android 应用具有不同数量级的可执行代码行 (ELOC),能够进

一步展示本文方法在不同复杂程度的 Android 应用程序上的测试表现。为了研究方法的可扩展性,本文方法直接对这些 Android 应用程序进行端到端的测试,无需针对每个应用进行特定调整。基于此基准,将本文方法与现有的 Android 应用自动化测试方法的代码覆盖率、故障暴露数以及测试效率 3 个指标进行深入比较。

此外,在实验评估中,为了评估本文方法在检测 Android 应用故障方面的有效性,收集并分析了控制台报告的系统级故障。需要注意的是,用户级故障并不一定引发系统级别的崩溃,这取决于应用的鲁棒性和输入验证机制。此外,不同的 Android 应用对于用户级故障可能有着不同的定义,使现有方法难以区分是否发生用户级故障。因此,本文主要关注那些会导致应用程序崩溃或异常行为的系统级故障。所有检测到的故障都经过人工复查和确认,以确保识别出的异常确实是实际存在的问题,而非误报。

为了评估 DAE 的有效性,选择了以下对比方法及其变体作为基准比较对象。1) Monkey (M)<sup>[2]</sup>, 通过与屏幕坐标的随机交互生成伪随机的用户事件流。为了确保实验的公平性,进一步优化了 Monkey,使其动作空间与 DAE 相同。2) Distributed-Monkey (D-M), 将 Monkey 拓展成分布式系统,多台设备同时对被测应用进行随机测试。3) Stoat (St)<sup>[5]</sup> 是一种基于模型的方法。它首先采用随机有限状态机模型来描述自动执行程序的行为,随后利用 MCMC 采样来指导模型的突变,并生成测试用例。4) Distributed-Stoat (D-St), 将 Stoat 拓展成分布式系统,多台设备同时对被测应用进行基于 Stoat 的测试。5) NoAggregation-DAE (NA-DAE), 是 DAE 在测试时不进行聚合步骤的变体。

对于所有的实验,为每种方法分配了相同的时间预算(即 60min)。为确保在执行复杂动作后页面完全加载,在每个动作后加入 5s 的等待间隔。在实验中,DAE 配置了 2 个智能体,每个“回合”的持续时间为 6 min。根据先前工作的设置,Stoat 方法的两个阶段默认各分配 30 min。为了保证公平性,D-M 和 D-St 的设备数量同样设定为 2 台。为了从统计学角度减小随机因素的影响,所有实验均重复进行了 5 次,并将平均数作为最后的结果。在参数设置方面,强化学习中的折扣因子系数  $\lambda$  统一设置为 0.96;对于动作选择策略,采用了  $\epsilon$ -greedy 策略,并设置  $\epsilon = 0.2$ 。为了保持一致性,所有测试方法的代码覆盖率计算均使用 Jacoco<sup>[23]</sup> 工具。对于分布式系统,利用 Jacoco 的 merge 命令将多台设备上的覆盖率数据合并,得到一个综合的覆盖率值,以此作为整个分布式系统的覆盖率。

### 4.2 实验结果与分析

表 1 列出了 M, D-M, St, D-St, NA-DAE 和 DAE 在 10 个开源应用上测试的平均指令覆盖率,其中最佳结果用粗体灰底表示。Android 应用根据从 Jacoco 覆盖率报告中提取的 ELOC 进行排序。总体而言,提出的方法在 8/10 个 Android 应用中取得了最佳代码覆盖率(并列第一也计入)。紧随其后的是 NA-DAE,在 5/10 的 Android 应用上取得最佳结果。从平均覆盖率来看,DAE 实现了 57.7% 的平均指令覆盖率,高于 M(41.1%), D-M(53.4%), St(46.8%), D-St(48.0%) 和

NA-DAE(56.6%)。

表 1 还列出了 M,D-M,St,D-St,NA-DAE 和 DAE 在 10 个开源应用上的故障检测数量,其中最佳结果用粗体灰底突出显示。总体而言,所提方法在 10/10 个 Android 应用上检

测到了最多的应用故障(并列第一也计入)。DAE 检测到了 16 个 Android 应用故障,高于 M(9 个),D-M(12 个),St(10 个),D-St(10 个)和 NA-DAE(15 个)。此外,已将所发现的故障报告给开发人员,并获得了部分确认。

表 1 测试结果的对比

Table 1 Comparison of testing results

Android 应用		代码覆盖率/%						Android 应用故障数量/个						
应用名称	功能	ELOC	M	D-M	St	D-St	NA-DAE	DAE	M	D-M	St	D-St	NA-DAE	DAE
budgetwatch	预算监督	1 349	66	<b>85</b>	74	74	<b>85</b>	<b>85</b>	0	0	0	0	0	0
account	开支明细	2 386	65	88	74	79	<b>90</b>	88	0	0	0	0	0	0
soundboard	音频播放	3 209	31	48	39	42	49	<b>50</b>	1	<b>2</b>	1	1	<b>2</b>	<b>2</b>
Sketches	画图	6 248	40	52	44	47	<b>59</b>	<b>59</b>	0	0	0	0	<b>1</b>	<b>1</b>
Calendar	日历	6 384	49	70	64	64	<b>72</b>	<b>72</b>	0	0	0	0	0	0
tally	账本分析	6 697	49	69	59	60	72	<b>75</b>	0	0	0	0	0	0
simplenote	备忘录	9 527	23	30	29	29	<b>34</b>	33	4	5	5	5	<b>6</b>	<b>6</b>
fillup	PDF 编辑	11 163	28	29	25	25	29	<b>33</b>	2	2	1	1	2	<b>3</b>
Keepsaddroid	密码管理	37 701	11	12	12	12	23	<b>26</b>	0	0	0	0	<b>1</b>	<b>1</b>
uhabits	习惯养成	42 146	49	51	48	48	53	<b>56</b>	2	<b>3</b>	<b>3</b>	<b>3</b>	<b>3</b>	<b>3</b>

为了评估测试效率,在 60 min 的测试时间内每 3min 收集一次代码覆盖率,绘制了每个 Android 应用在测试期间代码覆盖率的变化曲线图,如图 3 所示。图 3 为分布式系统(包括 D-M,D-St,NA-DAE 和 DAE)在 10 个应用中的代码覆盖率变化曲线图(横轴表示测试时间,纵轴表示代码覆盖率)。总体而言,DAE 方法在 Android 应用测试中具有最高的测试效率,其代码覆盖率增长曲线平均在 21.3min 内收敛,而 D-

M,D-St 和 NA-DAE 分别在 32.4 min,26.4 min 和 25.5 min 内收敛。DAE 的测试效率较 D-M,D-St,NA-DAE 分别提升 34.3%,19.3%,16.5%。这些结果表明,通过定期迭代聚合所有智能体累积的学习经验,DAE 充分利用了每个智能体的学习能力,从而显著提高了测试效率和代码覆盖率的生长速度。这不仅证明了 DAE 的有效性,也突显了其在自动化测试领域的优势。

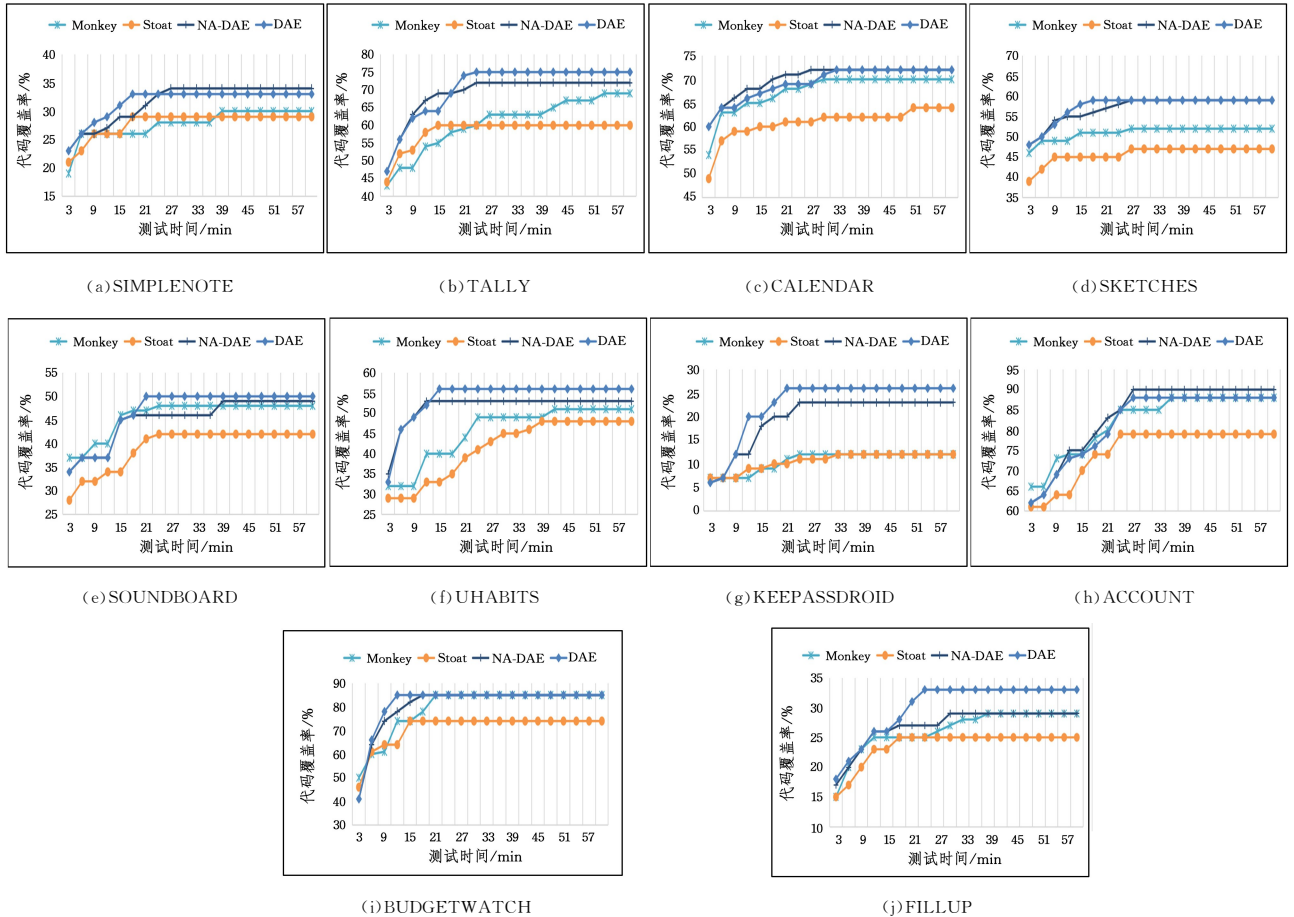


图 3 应用运行时的代码覆盖率变化  
Fig. 3 Code coverage changes at runtime

进一步对聚合步骤中智能体分发和同步管理维护造成的性能损失进行评估。在 60 min 的测试时间中, DAE 平均在聚合步骤消耗 1.67 min, 造成 2.8% 的性能损失。我们认为 DAE 造成的性能损失在可接受范围内。

## 5 相关工作

目前已经提出许多 Android 自动测试技术。基于随机的策略<sup>[2-4]</sup>随机生成 Android 应用程序的用户操作和输入, 旨在探索 Android 应用程序并发现潜在的故障和异常。例如, Monkey<sup>[2]</sup>通过与屏幕坐标的随机交互生成伪随机用户事件流, 然后进一步执行生成的用户事件流, 以方便对 AUT 进行测试。Paydar 等<sup>[3]</sup>开展了两项实验研究, 进一步考察了 Monkey 在识别 Android 应用的鲁棒性和响应性故障方面的有效性。然而, 随机策略无法保证全面的状态覆盖, 一些难以触发的隐藏功能可能永远不会被探索到。相比之下, DAE 采用强化学习算法实现了高覆盖率和高效测试。

基于模型的策略<sup>[5-8]</sup>采用动态或静态策略, 在测试前构建应用程序模型, 用于描述应用程序的行为。然后, 根据构建的模型生成测试用例, 对应用程序进行探索, 从而检测出任何存在的故障。例如, Stoa<sup>[5]</sup>利用加权 UI 探索策略, 精心设计了一个有限状态机作为测试应用程序的模型, 还利用 MCMC 采样对模型进行了迭代变异。此外, AndroidRipper<sup>[6]</sup>和 MobiGUITAR<sup>[7]</sup>利用以图形用户界面为中心的翻录技术进行应用程序探索。由于 Android 应用程序可能具有复杂的状态和行为, 因此很难构建完整的应用程序模型。如果模型不完整, 那么生成的测试用例的质量较低。相比之下, DAE 是一种基于强化学习的方法, 其测试不受模型限制。

系统方法<sup>[9-11]</sup>采用符号执行或进化算法等先进技术, 为目标应用行为提供特定输入, 旨在探索其他方法难以发现的复杂特征。Sapienz<sup>[9]</sup>基于帕累托最优多目标搜索方法, 旨在通过应用遗传、突变和其他运算符生成新的测试用例, 最大限度地提高代码覆盖率和增加故障暴露数量。然而, 这些方法的可扩展性较差, 在实际开发中的有效性也经常受到质疑。相比之下, DAE 可根据 Android 界面结构生成可执行操作, 缩短了探索无效操作的时间。

强化学习方法<sup>[12-16]</sup>利用智能体与真实环境(即 Android 应用程序)之间的交互所获得的奖励来修改探索策略。Q-testing<sup>[12]</sup>采用 Q-learning 方法, 根据当前应用状态与访问状态之间的相似度是否超过阈值来分配固定不变的奖励。ARES<sup>[13]</sup>采用深度强化学习, 根据状态的活动是否属于当前被测应用, 或者该活动是否已在测试过程中出现, 来分配固定不变的奖励。强化学习的奖励函数至关重要, 但现有的奖励函数仍显不足。我们的奖励函数考虑了实际状态之间的相似性, 它能准确估计每个状态的真实探索值, 防止探索过程陷入局部最优。

此外, 现有工作仅利用单台设备进行测试, 测试效率十分有限。DAE 利用多个智能体, 能同时对应用程序进行基于强化学习的测试, 并定期迭代地聚合所有智能体累积的学习经验, 充分利用了每个智能体的学习能力, 以此提高测试效率。

**结束语** 本文提出了一种基于强化学习的分布式 An-

droid 应用自动化测试框架 DistributedAndroidExplore, 利用多个智能体同时对应用程序进行基于强化学习的测试, 并定期迭代地交换累积的学习经验, 以此提高测试效率。实验表明, 在大多数情况下, DAE 的故障检测率、代码覆盖率都优于现有的先进的 Android 测试方法。同时测试效率明显优于其他方法, 性能提高了 16.5%~34.3%。

下一步研究将探讨当测试陷入局部最优时应如何跳出状态循环, 以进一步提升 Android 应用测试的效率。此外, 目前只使用 2 个智能体测试 10 个应用, 未来将使用更多的智能体对更多的应用进行进一步的测试和验证。

## 参考文献

- [1] TANG W H, TANG S H, LIAO C L. China's Mobile Internet in the 5G Era—Release of the “Development Report of China's Mobile Internet (2020)”[J]. China Newspaper Industry, 2020(17): 32-35.
- [2] ZELENCHUK D, ZELENCHUK D. Supervised monkey tests with espresso and UI automator[M]// Android Espresso Revealed: Writing Automated UI Tests. Berkeley, CA: Apress, 2019: 255-269.
- [3] PAYDAR S. An empirical study on the effectiveness of monkey testing for android applications[J]. Iranian Journal of Science and Technology, Transactions of Electrical Engineering, 2020, 44(2): 1013-1029.
- [4] KALYSCH A, DEUTEL M, MÜLLER T. Template-based Android inter process communication fuzzing[C]// Proceedings of the 15th International Conference on Availability, Reliability and Security. 2020: 1-6.
- [5] SU T, MENG G, CHEN Y, et al. Guided, stochastic model-based GUI testing of Android apps[C]// Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. 2017: 245-256.
- [6] AMALFITANO D, FASOLINO A R, TRAMONTANA P, et al. Using GUI ripping for automated testing of Android applications[C]// Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering. 2012: 258-261.
- [7] AMALFITANO D, FASOLINO A R, TRAMONTANA P, et al. MobiGUITAR: Automated model-based testing of mobile apps[J]. IEEE Software, 2014, 32(5): 53-59.
- [8] NGUYEN B N, ROBBINS B, BANERJEE I, et al. GUITAR: an innovative tool for automated testing of GUI-driven software [J]. Automated Software Engineering, 2014, 21: 65-105.
- [9] MAO K, HARMAN M, JIA Y. Sapienz: Multi-objective automated testing for android applications[C]// Proceedings of the 25th International Symposium on Software Testing and Analysis. 2016: 94-105.
- [10] MAHMOOD R, MIRZAEI N, MALEK S. Evodroid: Segmented evolutionary testing of android apps[C]// Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. 2014: 599-609.
- [11] GAO X, TAN S H, DONG Z, et al. Android testing via synthetic symbolic execution[C]// Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering.

2018;419-429.

- [12] PAN M, HUANG A, WANG G, et al. Reinforcement learning based curiosity-driven testing of Android applications[C]// Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis. 2020;153-164.
- [13] ROMDHANA A, MERLO A, CECCATO M, et al. Deep reinforcement learning for black-box testing of android apps[J]. ACM Transactions on Software Engineering and Methodology, 2022,31(4):1-29.
- [14] GAO Y, TAO C, GUO H, et al. A Deep Reinforcement Learning-Based Approach for Android GUI Testing[C]// Asia-Pacific Web(APWeb) and Web-Age Information Management(WAIM) Joint International Conference on Web and Big Data. Cham: Springer, 2023;262-276.
- [15] LÓPEZ Y P, COLONNA J G, SILVA E D A, et al. Q-funcT: A Reinforcement Learning Approach for Automated Black Box Functionality Testing[C]// 2022 IEEE 2nd International Conference on Software Engineering and Artificial Intelligence (SEAI). IEEE, 2022;119-123.
- [16] COLLINS E, NETO A, VINCENZI A, et al. Deep reinforcement learning based Android application GUI testing[C]// Proceedings of the XXXV Brazilian Symposium on Software Engineering. 2021;186-194.
- [17] LU P, YANG C Z, YOU Z J. DiLAD: A Distributed Layout Testing Framework for Android Applications[C]// 2020 the 3rd International Conference on Computing and Big Data. 2020;24-29.
- [18] BORGES JR N P, GÓMEZ M, ZELLER A. Guiding app testing with mined interaction models[C]// Proceedings of the 5th International Conference on Mobile Software Engineering and Systems. 2018;133-143.
- [19] KOROGLU Y, SEN A, MUSLU O, et al. QBE: QLearning-based exploration of android applications [C] // 2018 IEEE 11th International Conference on Software Testing, Verification and Validation(ICST). IEEE, 2018;105-115.
- [20] WANG Y, CHEN Y R, CHEN X, et al. Automating Release of Android APIs via Computational Reflection [J]. Computer Science, 2022,49(12):136-145.
- [21] ESKONEN J, KAHLES J, REIJONEN J. Automating gui testing with image-based deep reinforcement learning[C]// 2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems(ACSOS). IEEE, 2020;160-167.
- [22] Android UiAutomator2 Python Wrapper [EB/OL]. <https://github.com/openatx/uiautomator2>.
- [23] EclEmma. JaCoCo Java Code Coverage Library[EB/OL](2020). <https://www.eclemma.org/jacoco/index.html>.



**SONG Rirong**, born in 2000, postgraduate. His main research interest is Android automated testing.



**CHEN Xing**, born in 1985, Ph.D, professor, Ph.D supervisor, is a member of CCF (No. 35725M). His main research interests include software systems and engineering approaches for cloud and mobility. His current projects cover the topics

from self-adaptive software, computation offloading, model driven approach and so on. He has published over 100 journal and conference articles, including IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Mobile Computing, IEEE Transactions on Cloud Computing, IEEE Transactions on Industrial Informatics, etc.

(责任编辑:何杨)