软件学报 ISSN 1000-9825, CODEN RUXUEW
[doi: 10.13328/j.cnki.jos.007313] [CSTR: 32375.14.jos.007313]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

移动应用 GUI 测试自动生成技术综述*

王 博[1,2], 陈 冲[1,2], 邓 明[1,2], 董 震[3], 林友芳[1,2], 郝 丹[4,5]

[1](北京交通大学 计算机与信息技术学院, 北京 100044)

[2](交通数据分析与挖掘北京市重点实验室 (北京交通大学), 北京 100044)

[3](复旦大学 计算机科学与技术学院, 上海 200438)

[4](北京大学 计算机学院, 北京 100871)

[5](高可信软件技术教育部重点实验室 (北京大学), 北京 100871)

通信作者: 董震, E-mail: zhendong@fudan.edu.cn; 林友芳, E-mail: yflin@bjtu.edu.cn

摘　要: 移动应用是近 10 年来兴起的新型计算模式, 深刻地影响人民的生活方式. 移动应用主要以图形用户界面 (graphical user interface, GUI) 方式交互, 而对其进行人工测试需要消耗大量人力和物力. 为此, 研究者提出针对移动应用 GUI 的测试自动生成技术以提升测试效率并检测潜在缺陷. 收集了 145 篇相关论文, 系统地梳理、分析和总结现有工作. 提出了“测试生成器-测试环境”研究框架, 将该领域的研究按照所属模块进行分类. 特别地, 依据测试生成器所基于的方法, 将现有方法大致分为基于随机、基于启发式搜索、基于模型、基于机器学习和基于测试迁移这 5 个类别. 此外, 还从缺陷类别和测试动作等其他分类维度梳理现有方法. 收集了该领域中较有影响力的数据集和开源工具. 最后, 总结当前面临的挑战并展望未来的研究方向.

关键词: 软件测试; GUI 测试; 测试生成; 移动应用测试; 安卓应用

中图法分类号: TP311

中文引用格式: 王博, 陈冲, 邓明, 董震, 林友芳, 郝丹. 移动应用GUI测试自动生成技术综述. 软件学报. http://www.jos.org.cn/1000-9825/7313.htm

英文引用格式: Wang B, Chen C, Deng M, Dong Z, Lin YF, Hao D. Survey on Automated GUI Test Generation Techniques of Mobile Applications. Ruan Jian Xue Bao/Journal of Software (in Chinese). http://www.jos.org.cn/1000-9825/7313.htm

Survey on Automated GUI Test Generation Techniques of Mobile Applications

WANG Bo[1,2], CHEN Chong[1,2], DENG Ming[1,2], DONG Zhen[3], LIN You-Fang[1,2], HAO Dan[4,5]

[1](School of Computer and Information Technology, Beijing Jiaotong University, Beijing 100044, China)

[2](Beijing Key Lab of Traffic Data Analysis and Mining (Beijing Jiaotong University), Beijing 100044, China)

[3](School of Computer Science, Fudan University, Shanghai 200438, China)

[4](School of Computer Science, Peking University, Beijing 100871, China)

[5](Key Lab of High Confidence Software Technologies (Peking University), Ministry of Education, Beijing 100871, China)

Abstract: Mobile applications, a new computing mode that has emerged in the past decade, significantly impact people’s lifestyles. Mobile applications primarily interact through graphical user interfaces (GUIs) and conducting manual testing for them requires significant manpower and material resources. In response to this, researchers propose automated GUI test generation techniques for mobile applications to enhance testing efficiency and detect potential defects. This study collects 145 relevant papers and systematically sorts out, analyzes, and summarizes existing work. This study proposes a research framework called “Test Generator-Test Environment” to categorize research in this domain based on the modules to which it belongs. Particularly, this study classifies existing methods roughly into five categories according to the methods on which the test generator is based: random-based, heuristic-search-based, model-based, machine-

* 基金项目: 国家自然科学基金 (62202040)
收稿时间: 2023-11-11; 修改时间: 2024-06-21, 2024-08-28; 采用时间: 2024-10-30; jos 在线出版时间: 2025-03-19

learning-based, and test-migration-based approaches. Furthermore, this study analyzes and discusses existing methods from other classification dimensions, such as defect categories and test action categories. Additionally, influential datasets and open-source tools in this field are compiled. Finally, this study summarizes the current challenges and provides an outlook on future research directions.

Key words: software testing; graphical user interface (GUI) testing; test generation; mobile application testing; Android App

1 引 言

随着无线网络通信和移动互联网的快速发展, 移动应用程序 (mobile application, App) 已经成为人们生产和生活中不可缺少的一部分. 如同其他场景下的软件, 移动应用程序也不可避免地存在缺陷. 通过软件测试等手段及时发现缺陷是保障其质量的重要手段. 移动应用程序的输入主要来自图形用户界面 (graphical user interface, GUI), 存在着复杂的人机交互机制. 因此, 移动应用程序依赖于开发者人工编写测试. 然而, 当前移动应用程序的数量存在持续快速增长的趋势, 截至 2024 年 6 月, Google Play 应用商店中的 Android 应用数量已超过 355 万, Apple 应用商店中 IOS 应用数量超过 180 万. 巨大的应用数量和复杂的交互带来无法承受的测试开销. 为了缓解这一问题, 研究者提出了移动应用 GUI 测试自动生成技术.

移动应用 GUI 测试自动生成技术 (后文简称为“移动测试生成”) 不但面临传统测试生成技术中的缺乏测试预言 (Oracle)、路径爆炸和状态爆炸等问题, 还面临移动智能设备计算场景带来的新挑战. 特别地, 与传统桌面应用 GUI 测试生成技术相比, 移动应用程序存在设备多样性, 涉及触控和语音等多模态人机交互方式, 响应式布局, 具有摄像头和 GPS 等多种硬件, 依赖网络链接, 具有应用商店规范等差异. 这些差异为移动测试生成技术带来了新的需求和挑战, 使得原有技术很难直接迁移到移动应用的场景中. 因此, 该技术自诞生以来一直是工业界和学术界的研究热点与难点. 相关科技企业和研究所不断推出新的测试框架和工具. 在学术界内也吸引了包括谢涛、Yves Le Traon 和 Andreas Zeller 等多位 ACM/IEEE Fellow 在内的知名研究团队展开研究.

本文的写作目的是系统地梳理现有方法, 提出该领域中一般性的研究框架, 从不同角度对现有移动测试方法进行分类整理, 归纳总结方法之间的演化历程和发展趋势, 收集并评述现有工作常用的工具和数据集, 并评述当前面临的挑战以及未来可能的研究方向, 从而为相关领域的研究者提供移动测试生成领域的研究概况, 并提供不同的分析视角.

我们首先简要介绍相关论文的收集过程. 该领域在过去 10 年中新方法与工具不断被提出, 在软件工程等方向的重要国际会议和期刊上每年均有数量稳定的文章发表, 全球学者还举办了 MOBILESoft 等相关研讨会. 在收集论文的过程中, 我们首先以 (Android∨Mobile∨GUI)∧(Automated Testing∨Test Geneartion) 为主要关键词, 在 ACM-DL、IEEE-Explore、Google Scholar、CNKI 等国内外学术数据库中搜索相关论文并人工选出与本文主题相关的论文. 随后我们检索 CCF 推荐的国际会议/期刊的论文列表并收集主题相关的论文. 特别地, 我们重点关注发表于 ASE、FSE、ICSE、ISSTA、TSE、TOSEM 等软件工程领域会议和期刊的论文. 最后, 针对该领域的活跃学者, 我们浏览其 DBLP 或 Google Scholar 主页, 查找遗漏的重要文献. 由于相关方法对工业界具有很大的实用价值, 我们对于会议中一些非 research track 的论文予以保留, 尤其关注 industry track、new idea track 和 demonstration track 等相关专栏. 在选取过程中, 我们优先保留通过同行评议的论文. 由于该领域的快速发展, 尤其是随着大模型等新兴人工智能技术的兴起, 部分论文仅有 arXiv 预印本, 我们将根据论文内容和被引用情况酌情添加.

我们最终选取了 2012 年 1 月–2024 年 9 月之间的总计 145 篇论文, 其中绝大多数来自软件工程方向的高水平学术会议的主会和期刊. 表 1 系统整理了本文收录的相关科研工作的发表源分布, 以及文章在中国计算机协会评级 (CCF ranking) 中的分类情况. 我们尽量涵盖了近 10 年 CCF A 类会议及期刊中的全部相关工作, 并兼顾发表在 B 类和 C 类会议及期刊上的相关工作. 其中, 我们重点考虑以移动应用为主题的相关会议, 如 MOBILESoft 研讨会等. 由于本文是系统文献回顾, 而不是系统文献映射研究, 因此针对相关文献介绍主要通过评述方式展开.

表 1　收录文献发表源统计

发表源类型	发表源惯称	发表源全称	CCF评级	论文数
国际会议	ICSE	International Conference on Software Engineering	A	25
	ASE	International Conference on Automated Software Engineering	A	23
	FSE/ESEC	Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering	A	20
	ISSTA	International Symposium on Software Testing and Analysis	A	18
	OOPSLA	Conference on Object-Oriented Programming Systems, Languages, and Applications	A	2
	CHI	Conference on Human Factors in Computing Systems	A	3
	ICSME	International Conference on Software Maintenance and Evolution	B	6
	ISSRE	International Symposium on Software Reliability Engineering	B	2
	SANER	International Conference on Software Analysis, Evolution and Reengineering	B	2
	ICST	International Conference on Software Testing, Verification and Validation	C	8
	QRS	International Conference on Software Quality, Reliability and Security	C	3
	Internetware	The Asia-Pacific Symposium on Internetware	C	2
	EASE	International Conference on Evaluation and Assessment in Software Engineering	C	1
国际期刊	TSE	IEEE Transactions on Software Engineering	A	6
	TOSEM	ACM Transactions on Software Engineering and Methodology	A	5
	IST	Information and Software Technology	B	5
	ASEJ	Automated Software Engineering	B	2
	STVR	Software Testing, Verification and Reliability	B	1
	SCP	Science of Computer Programming	B	1
	SQJ	Software Quality Journal	C	1
中文期刊	软件学报	Journal of Software	中文A	7
	中国科学: 信息科学	SCIENTIA SINICA Informationis	中文A	1
	计算机研究与发展	Journal of Computer Research and Development	中文A	1

图 1 展示了 2012 年以来的每年论文发表数量. 根据相关论文发表趋势可见该领域发展较为迅速, 尤其是自 2017 年以来每年发表文章数量均超过 10 篇. 其中, 在 CCF 推荐列表中, A 类会议和期刊中每年都有稳定数量的论文, 这也从侧面反映出该研究领域十分活跃.

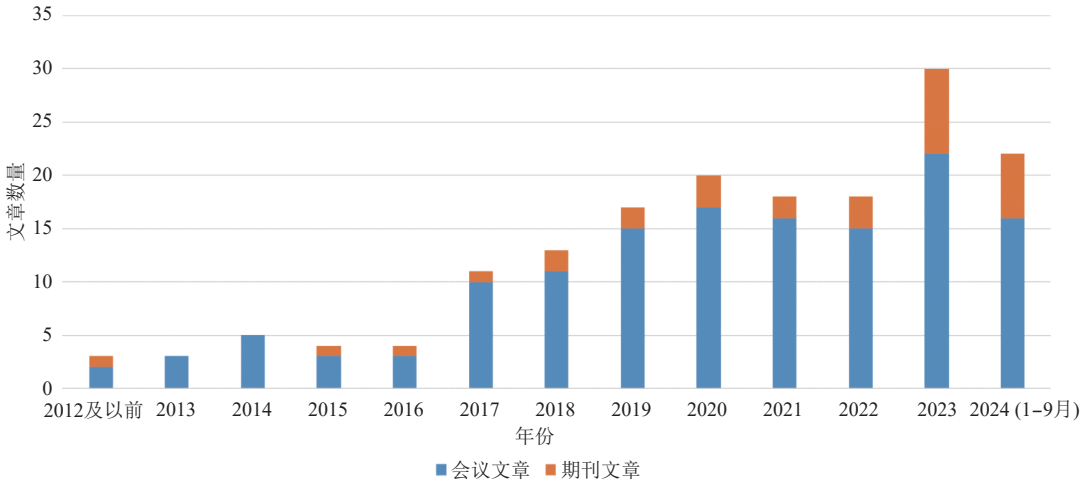


图 1　相关主题的文章按年份的分布

由于该领域已经有较长的发展时间, 已有国内外的多个研究团队对移动测试生成的相关问题进行综述. 王钰

等人[1]2019 年在《中国科学: 信息科学》上发表的中文综述, 针对该领域提出研究框架并对关键概念进行定义, 在此基础上汇总了 2019 年 1 月之前的文献. 同样, 在 2019 年, Kong 等人[2]在期刊《IEEE Transactions on Reliability》上发表的英文综述从多个角度系统性地梳理了 2010–2016 年间 Android 程序自动测试的研究, 并重点介绍了测试自动生成方法. Coppola 等人[3]于 2022 年发表在《Information and Software Technology》的回顾文章中总结了 GUI 测试中的覆盖度量标准, 但是没有从测试生成方法的角度综述. 除了上述综述以外, 该主题下也有系统映射研究的论文收集整理文献, 总结当前研究现状并描述面临的挑战[4–7].

在与该领域紧密相关的其他领域中, 国内外也存在一些综述工作. 李聪等人[8]于 2022 年发表在《软件学报》的综述中梳理了 Android 程序图形用户界面的录制重放测试技术. 录制重放技术的挑战是如何从人类众测数据中复原出相应的测试语义. 这类方法虽然能与测试自动生成技术形成互补, 但是在研究模型上存在较大区别. 郑炜等人[9]2022 年发表在《计算机研究与发展》上的综述介绍了安卓应用兼容性测试的相关工作, 与本文主要覆盖的功能性测试有较大区别. Luo 等人[10]于 2021 年发表在《ACM Computing Survey》的综述对测试过程中的上下文模拟技术进行综述. 这些方法是构建测试环境的重要环节, 但是并不主要涉及测试生成技术.

与这些已有综述相比, 我们的工作有如下不同.

首先, 本文以一个新的角度对移动测试生成领域进行梳理, 将移动测试生成任务抽象为一个新的研究框架并进行形式化定义. 具体而言, 本文将研究框架划分为测试生成器和测试环境, 分别进行建模, 并以测试生成方法为核心展开归类梳理. 相比于已有综述, 本文的研究框架考虑更加全面, 在形式化定义时考虑了系统配置和奖励估计方式等方面. 此外, 本文将已有测试生成方法分为基于随机、基于启发式搜索、基于模型、基于机器学习和基于测试迁移这 5 个类别. 相比于已有工作的分类, 本文的分类更加细致. 例如, 王钰等人的综述[1]将现有方法简单归类为基于遗传算法、基于机器学习和其他方法.

其次, 在上述综述之后, 各个类别中又诞生了新的代表性工作, 本文对此进行了补充. 例如, 在基于随机的方法中, 2021–2022 年间出现了一种通过重构用户界面布局提高随机试探命中率的方法, 显著提升了测试效率; 在基于启发式搜索的方法中, 近两年中出现了面向视障用户应用专门设计的搜索方法; 在基于模型的方法中, 自 2020 年后, 陆续出现了结合静态模型和蒙特卡洛树搜索生成探索动作的方法; 在基于测试迁移的方法中, 随着 NLP 技术的进步, 基于语义的组件匹配方法也取得了显著发展. 特别地, 本文重点补充了基于机器学习方法的相关工作. 随着深度学习技术的发展, 自 2019 年以来涌现出大量基于应用图像识别、自然语言处理、强化学习等技术的工作. 尤其是随着 ChatGPT 等预训练大语言模型的兴起, 在近期出现了一些基于大模型的移动测试生成技术. 我们的梳理工作也吸收了上述典型问题和代表性研究成果, 新增上述综述[1–3]未覆盖的参考文献 83 篇.

再次, 本文以移动平台上的应用测试自动生成为切入点. 与针对 Android 单一平台的文献[1,2,4]不同, 本文在选取工作的过程中并不限制移动操作系统, 在 Android 系统上的工作之外也补充了 iOS 等系统上的研究进展. 而与文献 [3,5] 等针对一般 GUI 测试生成的文献不同, 本文聚焦于分析移动平台应用交互特点, 并不涉及传统桌面应用的 GUI 测试.

最后, 相比于已有综述, 本文额外补充了与测试生成紧密相关的上下游技术, 并收集了影响力较大的数据集和工具. 部分研究论文在实验过程中会提出新的数据集, 本文根据文章的重要程度以及在后续研究中被使用的次数进行收集. 同时, 该领域的研究工作一般会发布新的测试生成工具, 并经常被未来研究工作当作基准比较对象. 本文收集了在实验验证中常出现的基准工具.

具体来说, 本文的贡献可以总结如下.

(1) 系统分析移动测试生成技术自诞生以来的发展历程, 收集并梳理了 145 篇国内外高水平学术论文.

(2) 提出了该领域的基于“测试生成器-测试环境”模型的研究框架.

(3) 在移动测试生成的研究框架下, 根据测试生成器中的生成方法的不同, 我们将现有方法分为基于随机、基于启发式搜索、基于模型、基于机器学习和基于测试迁移这 5 个类别.

(4) 在获得已有方法大致分类的基础上, 我们进一步分析不同方法类别的产生和发展趋势, 并从不同角度展开分析, 以便研究者快速掌握该领域的发展脉络和概况.

(5) 总结并分析了该领域常用的数据集和工具.

本文第 2 节建立移动测试生成技术的“测试生成器-测试环境”研究框架. 第 3 节梳理分析现有的移动测试生成方法, 主要从测试生成器角度下展开. 第 4 节总结相关研究中的重要测试数据集和开源工具. 第 5 节阐述该领域当前存在的挑战和未来研究方向. 第 6 节对全文进行总结.

2 移动应用 GUI 测试自动生成的研究框架

通过分析已有移动测试生成方法, 本文提出了“测试生成器-测试环境”研究框架, 如图 2 所示. 其中, 测试环境和测试生成器存在交互. 测试生成器向测试环境发送测试动作, 测试环境为测生成器返回当前环境状态和奖励. 测试环境是针对移动操作系统、系统和应用配置以及被测移动应用建立的模型. 该模型能够模拟操作系统执行应用程序, 并且能够分析和监控系统和应用程序的状态. 本文将测试环境的模型形式化定义为如下多元组:

$$M=(S,S_0,A,R,I,\delta,q).$$

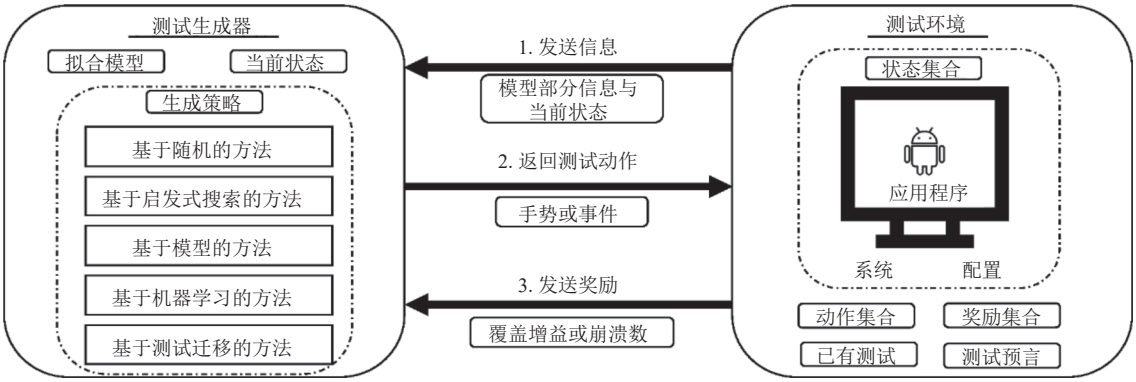


图 2 移动测试生成技术研究框架概览

测试环境的各个组成部分详细定义如下.

(1) 状态集合 $S=S_s\times C\times S_a$ 为状态空间, 即系统状态 S_s 与配置状态 C 以及应用自身状态 S_a 的笛卡尔积. 系统状态包括操作系统能访问到的所有资源的取值, 配置状态为硬件所有固有属性的取值, 应用状态为虚拟机等运行时视角内的资源的取值.

(2) 初始状态集合 $S_0\subseteq S$ 是模型所有合法初始状态构成的集合, 在本文中为测试执行之前的系统与应用状态.

(3) 动作集合 A 为动作空间, 即在给定系统和应用中用户所有可能实施的动作, 如点击屏幕或在文本框内输入文字等.

(4) 奖励集合 R 为奖励空间, 用于描述迁移到某状态后模型所提供的奖励, 例如代码覆盖率或程序崩溃次数等.

(5) 现有输入集合 I 为已有测试, 即给定初始状态和动作序列下的状态迁移过程集合. 在建模时该集合若为空集则表示无需使用已有测试的信息.

(6) 状态转移函数 $\delta:S\times A\to S$ 为在给定状态下实施某动作之后所迁移到的状态.

(7) 奖励函数 $q:S\times L\to R$, 用于描述模型从某初始状态执行动作序列后所获得收益. L 是动作序列集合, 即 $L=\left\{l|l\text{是有序序列}\wedge i\in\mathbb{N}\wedge l[i]\in A\right\}$.

为了更好地建立模型, 测试环境可以对被测应用和操作系统实施动态或静态分析, 并采用合适的抽象程度. 特别地, 测试环境可以在系统层面进行监控, 例如监视操作系统是否触发崩溃和异常等缺陷, 或者监视系统的内存、电力、网络等资源的消耗情况.

测试生成器是对测试生成方法集合的抽象, 它根据测试环境提供的信息, 评估当前状态下不同动作的收益, 完

成测试动作的生成并返回给测试环境. 本文将测试生成器形式化定义为如下多元组:

$$G=(M', s_{\text{curr}}, p).$$

测试生成器的各个组成部分详细定义如下.

(1) M' 是测试生成器根据测试环境提供的信息对模型 M 的拟合, 即 $M'=(S', S'_0, A', R', I', \delta', q')$. 其中, S', S'_0, A', R', I' 等集合分别为对应 M 中集合的子集 (例如, $S' \subseteq S$), 而函数 δ' 和 q' 分别为测试生成器的拟合.

(2) 当前状态 $s_{\text{curr}} \in S'$ 用于指向测试生成器 M' 当前所处状态.

(3) 策略函数 $p: S' \to A'$ 在给定环境状态下返回执行动作.

测试环境完成建模之后, 会将模型 M 中的部分信息和当前状态 s_{curr} 提供给测试生成器. 测试生成器根据测试环境所提供的信息和自身的生成策略, 根据对候选动作的收益估计, 生成测试动作并返回给测试环境. 测试环境根据模型的状态转移函数完成状态更新, 根据奖励函数计算所得收益并返回给测试生成器. 上述过程不断迭代, 直至达到时间、次数或者计算资源的限制. 最后由测试环境收集汇总得到自动生成的测试集. 对于移动应用界面测试而言, 我们希望在有限的时间和资源内尽可能多地获得奖励, 例如尽可能多地探索不同的屏幕状态或者触发更多的崩溃.

本文梳理相关工作的核心角度是测试生成器生成动作的方式. 根据生成动作的方式不同, 本文将现有工作分为基于随机、基于启发式搜索、基于模型、基于机器学习、基于测试迁移这 5 类方法. 目前, 主流测试生成技术由测试环境将应用程序的界面所有可能的情况抽象为有限的状态空间并传送给测试生成器, 再由测试生成器生成探索动作. 值得说明的是, 状态空间既可以是结构化的组件关系, 也可以是像素组成的应用运行时截图. 探索动作根据覆盖率增益或者触发崩溃等情况被赋予不同的奖励. 其中, 覆盖率可以分为代码层面、功能层面、状态层面和界面层面[6]. 其中, 代码层面的覆盖包括行覆盖、分支覆盖、路径覆盖、方法覆盖、类覆盖等; 功能层面的覆盖包括覆盖的故障、异常的数量和功能模块覆盖等; 状态层面的覆盖主要是将应用运行建模为状态转换图后统计状态覆盖率、事件覆盖率和跳转关系覆盖率, 界面层面的覆盖主要统计组件覆盖率和页面覆盖率.

目前, 现有移动应用界面测试相关工作对于测试生成器和测试环境都有相关研究. 一方面, 针对测试生成器的研究主要探索根据不同手段生成测试动作 (如输入文本框或者点击按钮). 在该领域中, 大多数论文针对生成方法展开研究. 如前文所述, 本文也按照生成方法维度将现有方法大致分为 5 类. 另一方面, 针对测试环境的研究主要从状态和奖励两个方面展开. 针对状态的研究主要是提出不同的状态抽象和系统建模的策略, 例如, 应用白盒分析还是黑盒分析, 返回界面控件组成关系还是界面源码甚至直接使用界面图像等. 针对奖励的研究则主要涉及挖掘新的测试预言和新的覆盖率计算方式.

3 移动应用 GUI 测试自动生成技术分类

本节主要在上一节提出的研究框架中分析梳理现有移动测试生成技术. 针对移动应用测试生成工具的分类学研究已有较多可参考的标准. 例如, Kong 等人[2]在 2018 年的研究中提出从测试目的、测试对象、测试层次和测试技术这 4 个维度对现有工作进行分类. 我们在充分考虑了用户界面测试任务的特殊性, 并结合我们在第 2 节中提出的研究框架之后, 在 Kong 等人的工作基础上, 提出从缺陷类别、测试生成的动作类别、测试动作序列的生成策略这 3 个主要维度出发, 对本文收录的测试生成器展开分类和整理. 值得注意的是, 本文的主要工作是从测试动作序列生成策略角度展开分析, 其他角度主要辅助介绍. 表 2 简要地展示了我们的分类框架.

表 2 移动应用 GUI 测试生成器分类框架

分类维度	子类别	代表测试工具
缺陷类别	用户接口缺陷	GLIB[11], Groundhog[12]
	用户交互缺陷	Odin[13], Route[14]
测试动作类别	非系统级事件动作	TOGGLE[15]
	系统级事件动作	Dynodroid[16]

表 2 移动应用 GUI 测试生成器分类框架 (续)

分类维度	子类别		代表测试工具
测试动作序列生成策略类别	基于随机		Monkey, EHBDroid[17]
	基于启发式	基于遗传算法	Sapienz[18]
		基于启发式符号执行	Columbus[19]
	基于模型	基于静态分析模型	A3E[20], SIG_Droid[21]
		基于动静态结合模型	APE[22]
	基于测试迁移	基于组件匹配	ATM[23], TestMig[24]
		基于语义提取	SemFinder[25]
	基于机器学习	基于强化学习	Q-testing[26]
		基于有监督学习	AppFlow[27], Avgust[28]
		基于预训练大语言模型	GPTDroid[29], QTypist[30]

我们同样考虑了其他主流分类维度, 例如基于测试阶段的划分. 但是综合分析调研结果后, 可以发现绝大部分面向用户界面的测试生成器集中在系统测试阶段工作, 仅少数工作涉及验收测试和回归测试[31,32]. 并且, 根据我们的调研, 没有工作涉及单元测试和配置项测试等. 我们认为移动应用 GUI 测试的特点不适用于该分类角度.

此外, 我们注意到部分测试生成工作面向产生新测试之外的其他任务, 如提升已有测试方法的测试效率、可用性测试等. 由于其数量较少, 我们同样并未将其作为一个独立的分类维度, 而是在相关工作介绍中对其单独进行系统整理.

3.1 缺陷类别

作为复杂系统, 移动 GUI 程序缺陷有多种不同的表现形式. 我们参考 Lelli 等人[33]和 Xiong 等人[34]针对 GUI 和移动应用程序等领域提出的缺陷分类框架, 将移动测试生成方法覆盖的缺陷类型分为用户接口缺陷和用户交互缺陷.

用户接口缺陷主要包括 GUI 架构与美学相关错误和数据呈现相关错误. 前者主要涉及控件的布局、显示状态和外观等错误, 后者主要涉及数据渲染和数据展示形式等错误.

用户交互缺陷指当系统外部 (用户或者测试生成器) 提供的动作应用在移动系统或者应用上时, 没有按预期改变系统或者应用状态而产生的缺陷. 用户交互缺陷可以分为交互行为错误、操作错误、操作撤销错误和反馈错误等. 该类缺陷与软件的功能正确性和安全性等重要属性相关, 是当前已有研究的重点. 然而, 针对用户交互缺陷的测试面临预言不足的问题, 当前已有工作主要以检测崩溃缺陷为主.

3.1.1 面向用户接口缺陷的测试生成

显示缺陷主要出现在页面布局与渲染复杂且渲染负荷较高、图像组件丰富的应用程序中. Xiong 等人[34]将用户接口缺陷分为目录显示问题和 UI 结构显示问题 (如多余组件和布局混乱等). Chen 等人[11]通过对手机游戏的显示缺陷进行研究, 基于组件类别给出了 8 种更加详细的分类标准, 并进一步分析了其根因. 他们将收集的显示缺陷作为训练集, 使用基于图卷积网络的分类模型实现了针对游戏类应用显示状态缺陷的有效修复. Liu 等人[35]收集来自百度众测平台上的 6 729 张具有用户界面显示缺陷的应用程序截图和 GitHub 中 1 016 个缺陷报告提供的应用程序截图, 采用主题分析方法识别出 9 类用户界面显示缺陷, 并据此总结出了界面显示缺陷的根因. Li 等人[36]关注界面图片显示相关缺陷, 他们收集了 162 个真实图像显示缺陷并通过分析其特征提出了缺陷检测工具 TAPIR. 该工具成功发现了 43 个未知的图像显示缺陷.

除了直接影响使用的显示故障外, 面向界面显示的易用性测试也是当下较为重要的一个研究方向. 设计不合理的 UI 界面会增加应用的使用难度. 例如, 过低的界面配色对比度会导致屏幕模糊, 组件布局不合理会导致点击不便等. Eler 等人[37]总结了 5 种界面显示相关的可用性错误, 包括带有自动阅读功能文本的可用性、界面配色的对比度、触摸组件的判定范围大小、组件的点击判定框的重叠情况、可点击文本的可用性等. Salehnamadi 等人[12]

研究了应用界面布局对视觉障碍辅助工具 TalkBack 的支持能力. Salehnamadi 等人[38] 实现了基于录制重放技术的可用性测试方法. 该方法记录用户使用常规交互方法完成某测试用例的动作序列, 并尝试调用各种辅助交互工具执行上述动作, 并生成可视化的测试报告. Alshayban 等人[39]则针对移动应用程序针对弱视人群提供的文本缩放辅助服务中存在的不兼容问题进行测试.

由于移动应用中丰富的控件和多模态展示方式, 用户接口缺陷存在丰富的类型和表现形式. 目前, 针对用户接口缺陷的测试主要依赖人工. 自动测试工具主要集中在解决某一特定类型应用中有限类型的显示缺陷. 针对用户接口缺陷的测试生成研究仍有较大的发展空间.

3.1.2 面向用户交互缺陷的测试生成

检测用户交互性缺陷的核心是提供足够的测试预言, 即通过比较执行组件交互动作的预期结果与实际结果来判断功能是否存在缺陷. 目前为止, 大部分测试生成方法只支持使用程序崩溃、资源泄漏等被动预言作为测试预言, 只有少量研究涉及主动生成测试预言, 即显式地检查条件验证程序正确性. 因此, 可以根据测试生成器是否支持主动生成预言进行分类. 本节主要介绍支持主动生成预言的代表性工作.

挖掘动作与语义之间的蜕变关系是主动构造测试预言的一种有效策略. Zaeem 等人[40]总结出了放大、缩小、滚动、 旋转等手势对应的状态转换规则. 这些特定手势在不同应用中对应的语义类似, 因而可以用相同的规则描述. 但是该方法对于点击、长按等不存在固定语义的交互动作无效.

Mariani 等人[41]提出从应用无关的功能中挖掘测试预言. 常见的该类功能包括身份验证操作、数据的增删改查操作、搜索和预定操作等. 若挖掘出其语义正确性条件, 则可以在其他应用上重用.

Baral 等人[42]通过调研测试报告的方式对移动应用常见的功能性缺陷类型进行分类研究, 并人工总结了若干应用无关的功能性测试预言.

2021 年, Su 等人[43]挖掘出新的蜕变关系, 用于增强基于模型的测试生成方法的预言. 他们首先提出了视图独立属性, 即运行与某一视图无关的视图不会影响该视图的状态, 并利用视图独立属性作为测试预言来设计测试生成方法. 在构建移动应用运行模型后, 给定某一活跃视图, 从其无关的视图出发来生成事件序列, 并最终返回到当前视图, 继续执行给定的活跃视图. 通过比较直接执行给定活跃示图和先执行构造的事件序列在执行给定活跃视图的状态是否相同来检测功能缺陷.

Wang 等人[13]在 2022 年提出了另一种针对功能缺陷的测试生成方法 Odin, 通过使用深度状态差分分析 (deep state differential analysis) 作为测试预言. 该预言基于的假设是在自动生成的测试中有大量输入最终会到达相似的界面, 并且在相似界面上执行同样的事件只会让应用产生有限的行为. 因此, Odin 给到达相似界面的测试输入添加相同的事件序列, 将应用所产生的行为中的多数作为测试预言, 并将少数行为视为潜在的功能缺陷.

2024 年, Xiong 等人[44]又提出使用基于属性的测试方法挖掘测试预言, 并实现了测试工具 DroidChecker. 该方法首先定义一种属性描述语言 (PDL) 以形式化地提取并记录应用界面特征, 并在此基础上设计了两种探索策略以快速地遍历并检查这些特征是否被违背. 实验证明基于上述两种探索策略, DroidChecker 能够发现并检测出绝大部分已知历史缺陷.

另一种主动生成预言的方式是复用或部分复用已有人工编写的测试预言. 代表工作有 Lin 等人[14]在 2022 年提出的 Route. 其能够在接收待测移动应用及其中某一个功能的测试脚本后自动查找并补全到达相同界面的其他可能的交互动作路径, 并产生相应的测试脚本. 这样产生的脚本预期功能与原测试一致, 因而可以直接复用输入的原始脚本中的测试预言.

3.2 测试动作

根据测试生成器能够向测试环境发送的测试动作种类的区别, 可将相关工作分为非系统级事件生成和支持系统级事件生成的方法. 测试生成器能够产生的动作种类对测试环境的状态集合 S 的大小有直接影响. 由 S 的定义 $S = S_s \times C \times S_a$ 可知, 状态集合包含的状态数受系统状态数 S_s、配置状态数 C 以及应用自身状态数 S_a 的综合影响. 绝大部分测试生成器能够实现遍历应用自身状态集合 S_a, 但仅有支持系统及事件生成的测试生成器可以实现

遍历系统状态 S_s, 因而该类方法对应的测试环境中状态集合包含的状态数更多, 容易面临搜索路径爆炸等问题, 实现难度也更大.

常见的能够对应用界面转移产生影响的系统级事件包括切换屏幕、接打电话、连接服务器、拉起支付程序等. 移动应用平台系统事件数量庞大, 除了引入系统事件将大幅扩大被测应用状态集合数外, 某些特定类型的缺陷也仅会由系统事件级别的交互动作引发, 如由于系统事件打断, 导致用户输入的信息丢失等. 综上所述, 实现测试生成器对系统级事件的支持是一个重要的研究方向.

3.2.1 非系统级事件动作生成

根据交互组件类型划分, 非系统事件主要包括文本输入类事件和非文本输入类事件 (包括点击、拖动、长按等交互手势). 对于文本输入事件, 测试生成器需要通过有限的测试输入达到覆盖全部候选转移界面的目的, 通常通过自然语言处理相关技术来实现. 对于非文本输入事件, 测试生成器需要首先定位待测组件, 并基于预设的状态空间探索策略生成交互动作序列. 我们将在第 3.3 节中主要介绍常见的状态空间探索策略, 本节介绍主流的 UI 组件定位方法.

当前主流的组件定位方法包括基于布局的方法和基于可视化的方法. 基于可视化的定位方法通过图像识别技术捕捉 UI 组件, 相较于基于布局的方法具有跨平台、跨系统、跨编程语言等优势, 但存在对 UI 布局微调敏感的缺陷, 需要不断地人工维护以适应应用的更新迭代. Ardito 等人[45]通过邀请研究生使用对应工具为开源应用编写测试, 比较了这两种方法的定位效果, 发现被试者在两个工具上的生产力表现相近, 但基于可视化的定位工具写出的测试质量更好. Coppola 等人[15]提出的 TOGGLE 实现了自动地将基于布局的定位方法编写的测试套件转为基于可视化的测试套件的过程, 解决了可视化测试套件需要频繁维护的问题. 2021 年, 该团队[46]在 TOGGLE 的基础上增加了支持基于上下文的手势以及支持基于布局的定位器与逻辑运算符的组合来对其进行扩展, 显著增加了其可转化的测试套件比例.

Coppola 等人[47]同样探讨了基于图形的测试套件转为基于组件的自动转化框架的可行性. 经过研究, 他们将这一过程分为 4 个阶段: 应用源码插桩 (植入定位函数的回调函数); 执行基于图像的方法的测试脚本并收集交互日志; 解析日志, 生成基于组件的脚本.

整体而言, 定位图形界面组件是测试生成工具产生探索动作序列的必要步骤. 是生成非文本类交互事件方法实现过程中尤为重要的一环. 由上述工作可见该领域受到研究者的持续关注, 陆续有新方法被提出.

3.2.2 系统事件动作生成

针对系统级事件的缺陷检测能够通过静态分析方法实现. 例如 2023 年, Yang 等人[48]提出的 PSDroid 使用传统的路径敏感的语义分析方法, 实现了高效且自动的 API 调用完整性错误检测. 相比于静态分析方法, 测试生成方法相对不依赖对 SDK 框架源码的理解, 具有较好的可移植性, 且不易受到移动端软硬件版本更新所带来的影响.

2013 年, Machiry 等人[16]提出的随机测试工具 Dynodroid 是首个将系统级事件纳入测试生成工具能够生成的探索动作集合中的尝试. 为了回避引入过多系统事件导致搜索空间爆炸, Dynodroid 只记录应用注册的系统事件. Auer 等人[49]进一步研究了平衡测试输入中非系统事件与系统事件输入的占比的问题, 并据此提出了基于模糊测试的意图测试自动生成框架, 将交互事件和意图的自动生成有机结合并应用于测试生成.

如上文所述, 引入系统级事件生成能够发现更多的缺陷类型, 例如数据丢失问题. 该问题表现为用户切屏或接电话等事件介入打断当前界面活动, 并导致用户返回应用界面时, 已执行的操作或已输入的信息丢失. 若干研究者针对这一问题给出了相应的检测手段. Guo 等人[50]提出的 iFixDataLoss 通过静态数据流分析工具提取状态转换图来引导测试, 在每一个节点使用点击返回按钮、关闭应用和旋转屏幕这 3 种外部事件进行测试. Rahaman 等人通过分析事件处理函数导出组件对应的需要重新载入的数据的二分图, 并基于二分图构造数据流. 其中, 包含不完整或错误的存储-重新载入关系的数据流代表一个数据丢失错误[51].

除了通用系统级事件生成方法外, 某些相关研究针对特定类型的系统事件引发的故障. Fan 等人[52]研究了移动应用界面线程与系统级线程的异步交互引发的相关故障. Ravelo-Méndez 等人[53]提出了针对包含跨设备信息交互功能 (如聊天功能等) 的用户界面测试框架. Su 等人[54]还针对系统设置相关用户界面缺陷进行了相关研究. 系统

设置是一类特殊的系统级事件, 包含切换飞行模式和低功耗模式或者横屏模式等. 该类缺陷在 Android 应用中普遍存在, 且 70% 以上的设置相关缺陷不会触发崩溃, 而是功能性缺陷. Su 等人在此发现的基础上提出的测试生成工具在抖音和微信等移动应用上成功地发现了缺陷.

尽管该领域已有上文所列代表性工作, 当前支持系统级事件生成的测试生成器相关研究整体数量仍然相对较少. 如何处理大量多种系统事件, 引入更多缺陷类型和更加复杂的界面转移关系是该方向下测试生成器相关研究当前面临的一个主要问题.

3.3 测试动作序列生成策略类别

从测试生成器采用的探索动作生成策略函数的角度, 已有生成方法可被分为基于随机、基于启发式搜索、基于模型、基于机器学习、基于测试迁移这 5 类. 对于测试生成器而言, 其目标是根据测试环境提供的状态信息生成探索动作序列, 通过产生连续的探索动作实现在不同屏幕状态之间转移, 尽可能多地探索不同的屏幕状态. 一般而言, 给定一个屏幕状态, 可选的探索动作空间是有限的, 测试生成器需要在考虑当前和未来收益的情况下尽量选择最优的动作. 在本节中, 本文首先按照相关工作的出现顺序依次介绍 5 类生成方法中的代表性工作, 并在本节最后对该领域的发展演化趋势进行小结.

3.3.1 基于随机的方法

基于随机的测试生成器的策略函数在给定当前状态下可采取的动作中随机选取一个进行执行. 在选择探索动作时, 该策略既不考虑应用上下文信息, 也不区分不同探索动作对产生新状态的影响, 总是以相同的可能性从备选探索动作中选择一个完成探索. 虽然理论上随机测试通过足够多次数的探索能覆盖目标应用中大多数的状态转移路径, 但在实际应用中往往容易遭遇覆盖瓶颈, 即执行一段时间后覆盖不再增加的情况.

基于随机的测试生成方法中最具代表性的工具是 Android 应用开发工具箱中的 Monkey. 该工具首先获取一个目标应用支持的交互动作集合 A, 其中包含上下滑动、点击、拖动、放大、缩小等常用手势动作. Monkey 通过以下两步生成一个探索动作: (1) 在当前界面上随机定位一个图形界面组件; (2) 在 A 中随机选择一个交互动作实施. 如果屏幕状态不发生变化, 则重新选择其他动作, 直到屏幕状态变化, 就完成了一次探索动作.

Monkey 的探索动作生成策略较为简单, 只需提供目标应用和交互动作集合即可进行测试. Monkey 生成的测试能正确运行的概率很高, 这使得 Monkey 在结构复杂的工业级应用上的测试表现反而优于一些繁琐的测试方法. 复杂的方法往往需要分析应用的界面结构, 静态分析过程中会导致误报和错误, 进而影响生成结果. Wang 等人[55]在针对主流自动移动测试生成工具展开的研究中, 发现 Monkey 在平均代码覆盖率和事件覆盖率上都优于其他自动测试工具.

Monkey 在存在诸多优点的同时也存在一定的不足. 对于移动应用, 屏幕状态转移除了受到应用自身探索动作影响之外, 还受到诸如点击系统返回键等系统级事件的影响. 然而, Monkey 不能产生系统级事件. 同时, Monkey 测试效果在不同被测应用上不能始终保持高质量. Zheng 等人[56]在微信上对 Monkey 进行测试, 发现其仅能达到 19.5% 的代码覆盖和 10.7% 的事件覆盖, 这证明 Monkey 在某些特定应用的测试中表现不稳定.

2013 年, Machiry 等人[16]提出了新的随机测试方法 Dynodroid, 实现了对系统级事件生成的支持. 为了不改动被测程序, Dynodroid 对 SDK 插桩来记录系统事件. 相比于 Monkey, Dynodroid 仅使用 1/20 的事件数目就能达到最高覆盖.

Dynodroid 得到了广泛关注, 但在研究者试图对 Dynodroid 进行复现时, 发现 Dynodroid 已经无法在最新版本的 Android 系统下启动. Borges 等人[57]在这一状况的启发下, 提出了开源的、可扩展的随机移动测试生成工具 DroidMate. DroidMate 相较于 Dynodroid, 结构更简单, 更新维护更加方便. DroidMate 在开源和工业应用测试中都有极高的生成成功率.

尽管陆续出现了其他随机测试工具, 但 Moneky 依然是最流行的工具之一. 在对 Monkey 的使用过程中, 研究者们提出了多种优化方法以提升其测试效率. Yan 等人[58]对 Monkey 测试中的事件有效性进行分析, 发现了 3 种典型的无效事件, 即无操作、单个无效事件、多个无效事件的组合, 并据此提出了 9 个约减规则. Behrang 等人[59]

分析了 Monkey 在 64 个应用程序上的性能, 并确定了 7 种一般类别的限制. Hu 等人[60]使用 Deeplink 标记关键页面引导 Monkey 的搜索过程, 从而避免陷入无意义的动作循环. 这些优化方法在保留 Monkey 随机选择探索动作机制的同时, 通过人工设置的约减规则对备选探索动作集合进行剪枝, 排除了明显对发现新状态没有帮助的探索动作.

Paydar 等人[17]提出了一种针对 Monkey 测试效率优化的新思路, 即通过重构界面布局本身, 而非优化测试工具来提高测试效果. 为了方便用户点击, 移动应用界面设计时遵循组件稀疏且大小相对一致的原则. 然而, 对 Monkey 而言, 理想的界面布局是组件分布稠密且间距小, 从而减少 Monkey 点击到屏幕空白处的概率. 同时, 更复杂的响应事件对应更大尺寸的组件, 使其能够以更大的概率被 Monkey 选中.

除了单独使用随机测试方式生成测试动作之外, 另一种基于随机的测试生成方案是将随机测试与人工测试相结合. 以人工编写的测试框架为主体, 并用随机产生的探索动作替代原有的部分探索动作, 生成新的测试脚本. 这种方法最先被应用于回归测试中. Wetzlmaier 等人[32]提出了基于人工测试脚本和 Moneky 的混合策略的回归测试思路. 他们分别在原有测试脚本的每个 checkpoint 处使用 Monkey 生成的交互序列代替该 checkpoint 之后的原有事件, 从而产生不同程度复用原脚本的新测试脚本.

在针对随机测试工具测试效果的评估方面, 同样陆续有研究者提出过一系列衡量指标. Godboley 等人[61]提出了两个针对敏捷式开发的移动应用的随机测试效果指标, 即元素覆盖率和处理函数覆盖率. 他们扩展了黑盒随机测试工具 Appium, 使其能够自动计算这两个指标并输出相关报告. 实验证明该方法能够有效加速敏捷开发式应用的界面测试效率.

随机测试生成工具的优势有以下几点: (1) 配置和启动非常简单, 开发者几乎不需要任何额外的学习即可使用; (2) 对于绝大多数目标应用都能成功产生测试; (3) 方法结构简单, 能够适应软硬件环境的更新迭代, 执行效率高. 但随机生成探索动作本身存在一些挑战. 例如, 应用的 GUI 状态转移结构通常不是均匀分布的, 有一部分屏幕状态连通更多其他状态, 而另一部分状态相对孤立. 由于随机选择探索动作不会感知当前屏幕状态信息, 它无法辨别探索动作的优劣, 难以有效扩展到更广泛复杂的状态空间, 从而限制了整体覆盖率的提升.

3.3.2 基于启发式搜索的方法

基于启发式搜索的测试生成器对应的探索策略使用研究者预定义的启发函数, 利用已有状态和动作的反馈信息引导后续探索动作的选择. 如前文所述, 随机选择探索动作在覆盖率上存在瓶颈主要因为其无法分辨不同探索动作的优劣, 从而投入大量资源尝试重复或无效的探索, 进而产生大量低质量测试. 为了解决这一问题, 研究者提出优化探索动作选择步骤的方案, 旨在每次生成探索动作都尽可能提高覆盖. 根据历史探索信息和对应用程序的分析, 测试生成器可以估计每一个状态下特定探索动作的潜在收益. 测试生成器通过启发式搜索, 对每个状态下不同探索动作进行收益估算, 并在测试生成过程中优先选择高收益的探索动作.

一般的启发式搜索任务的收益估算公式如下所示:

$$f(x) = g(x) + \hat{h}(x)$$

其中, $f(x)$ 代表从初始状态到某个最终状态的完整路径收益, $g(x)$ 代表初始节点到当前节点已经产生的收益, $\hat{h}(x)$ 代表当前节点的预期收益. 对于图形界面状态空间探索任务而言, $f(x)$ 代表一个完整的测试脚本的屏幕状态覆盖数, $g(x)$ 代表脚本执行到当前屏幕状态时的状态覆盖数, $\hat{h}(x)$ 则代表由当前状态出发到测试结束之间可能的最大状态覆盖数. $\hat{h}(x)$ 的设计是启发式搜索的关键, 必须准确地反映不同节点的收益大小关系. 具体到局部探索动作选择, 测试生成过程应该对当前状态每个备选的动作都进行评估, 并选择最大估计收益作为 $\hat{h}(x)$, 同时执行其对应的动作.

常见的基于启发式搜索的测试生成方法包括基于遗传算法的生成方法和基于启发式估值函数与符号执行技术相结合的方法.

对于基于遗传算法的方法而言, 测试生成器首先接收已有测试中的探索动作序列作为输入, 通过选择、交叉、变异等操作在输入序列的基础上产生子代序列, 计算评估函数值, 保留评估值最高的动作, 迭代地产生下一轮子代

和筛选高评估值的个体, 使搜索过程向评估值趋于更高的方向进行, 直到子代序列评估值趋于收敛. 因为变异操作无法感知相邻探索动作之间的内在联系, 完全基于遗传算法的搜索方式存在较多局限性. 现有的基于遗传算法的测试工具更多地采用遗传算法和随机搜索交替进行的混合搜索策略.

2016 年, Mao 等人[18]提出了面向移动应用的启发式图形界面测试自动生成工具 Sapienz. Sapienz 相较于早期 PC 端的 GUI 自动测试工具 EXSYST[62]有显著改进. 它初始种群的来源与 EXSYST 存在显著不同, 即首先检测待测应用源代码是否可获取. 如果可获取, 则通过解析源码得出一些与待测应用功能结构密切相关的测试作为初始种群. 在交叉和变异阶段, Sapienz 采用遗传算法和随机事件选取交替执行的策略, 其中随机事件选取主要负责遗传算法通常难以覆盖的转移操作, 比如需要输入一系列复合事件才能触发的界面转移关系, 从而解决了 EXSYST 子代相似性过大的问题. 在具体变异细节上, Sapienz 将单个图形界面事件作为变异单位, 允许变异在动作序列的任意位置进行. Sapienz 支持将明显具有逻辑依赖性的动作组合 (比如编辑文本和点击提交按钮) 作为一个不可分割的变异单位看待. 但为了防止引入过多的人为因素导致搜索出现偏见, Sapienz 对于此类组合事件的使用较少.

在评估函数的设计阶段, Sapienz 创新性地引入了基于帕累托最优的多目标优化方法. 定义 $f1(x)=\{$动作序列x的状态覆盖率$\}$, $f2(x)=\{$序列x包含的动作数量$\}$, $f3(x)=\{x$引发的程序异常次数$\}$, 则利用多目标优化模型描述的评估函数选择过程如下:

$$\begin{cases} V\text{-}min: f(x)=[f_1(x),f_2(x),f_3(x)]^{\mathrm{T}} \\ \text{s.t. } x\in X, X\subseteq R^m \end{cases},$$

其中, R^m 代表经由遗传算法得到的全部子代动作序列. 根据帕累托最优概念, 构造偏序关系. 当且仅当一个状态 s 在 $f1, f2, f3$ 上的值均优于另一个状态 s', 则认为 $s>s'$. 每经历一次迭代, Sapienz 将保留所有不存在大于它的个体的序列, 从而实现个体择优.

Sapienz 采用的遗传算法和随机生成交替的状态空间搜索策略, 以及基于帕累托最优的评估函数都对后续的相关研究产生了较大影响. 凭借着诸多新方法的提出和在大量真实应用上优秀的测试生成表现, Sapienz 成为该分类下的典型方法.

Vogel 等人[63]在 Sapienz 的基础上采用适应性地形分析理论进行优化, 提出了 Sapienz 的改进工具 Sapienz-div. 他们指出 Sapienz 基于 NSGA-2 遗传算法产生的后代在经过多次迭代后, 在状态空间中的相似距离分布过近, 存在多样性衰减的问题. Sapienz-div 分别从初始种群选取、遗传迭代、子代个体选择这 3 个阶段提出了扩展策略. Mao 等人[64]进一步将 Sapienz 与人群测试相结合, 通过人群测试提供的用户经验信息帮助 Sapienz 克服了缺乏对组件和动作语义信息的感知能力的弱点. Dong 等人[65]提出了基于可捕捉可恢复的状态进行演化的 TimeMachine 方法. 该方法能够系统性地重置 Android 系统到最有可能发现新状态的状态. 一旦测试被卡住, 系统就退回状态并重新开始探索. 在代码覆盖率和发现错误数量上, TimeMachine 显著优于 Sapienz.

除了变异和选择过程外, 另一个遗传算法相关的研究方向是种群划分方式. Mahmood 等人指出, 之前的遗传算法将完整的动作序列作为个体看待, 使变异可以在任意位置进行, 某些变异操作将亲代个体中的一些具有更强连通性的重要节点因变异而丢失, 导致子代的状态覆盖潜力骤降. 他们提出的 Evodroid[66]将完整的动作序列切分为较多更短的动作片段, 将这些片段作为遗传算法的个体使用. 在每个独立的个体中, 能够成功地从一个关键事件转移至另一个关键事件的子代将被赋予更高的估算权重.

以 Sapienz 为代表的遗传算法类探索动作评估方法实现较为简单, 不依赖于具体的图形界面布局结构, 具有较好的通用性. 但是, 该方法生成的测试质量依赖初始种群的选取. EXSYST 采用随机的初始动作序列, 虽然能给予探索动作平等的机会, 但不能保证初始序列的状态覆盖率, 需要多次迭代才能收敛. Sapienz 及其他一些主流方法采用已有测试脚本或从应用导出的启发式归约作为初始序列, 初始测试的质量更好, 但初始种群的选择相比 EXSYST 比较局限, 生成的子代可能缺乏差异性.

启发式算法在移动测试生成领域中时常会与基于符号执行技术的状态空间探索相结合, 通过启发式规则简化搜索空间, 达到降低复杂度的目的. 符号执行通过使用抽象的符号代替具体值来模拟程序的执行, 当遇到分支语句

时, 它会探索每一个分支, 将分支条件加入相应的路径约束中. 若约束可解, 则说明该路径是可达的. 符号执行能够避免生成重复的探索路径.

使用符号执行技术进行图形界面测试的关键在于平衡效率与成本. 真实应用中的图形界面的结构通常较为复杂, 包含大量逻辑分支, 而符号执行存在路径爆炸等问题, 在该领域上的应用受到了限制. 为此, 研究者提出一些适配方法以更好地应用符号执行技术.

Cheng 等人[67]提出将针对整个用户界面源码的符号演算任务切分为若干可以并行执行的分任务, 并通过高性能云服务器提高符号演算效率. 该方法生成的脚本中不仅包含普通的点击动作等事件, 还包含高质量的包含文本输入的事件.

除了从硬件计算资源层面解决之外, 另一种解决方法是从算法层面减少符号执行的开销. 符号执行领域常用的剪枝策略包括启发式的符号执行, 即优先探索最值得探索的路径, 并使用合理的程序分析技术来降低路径探索的复杂性. 还有动态符号执行, 即混合符号执行和具体数据执行的遍历方式.

Salvesen 等人[68]在 EXSYST 的基础上, 通过保留其基于遗传算法事件序列产生模块的同时, 引入了动态符号执行技术来填充测试过程中出现的输入文本框等组件, 有机地结合了遗传算法和符号执行技术.

Bose 等人[19]尝试将符号执行应用于基于事件处理函数的测试生成过程. 该方法通过绕过与屏幕组件的交互, 直接调用事件处理函数来测试, 在测试需要复杂交互动作触发的功能时具有优势.

表 3 总结了本节介绍的几种代表性的启发式工具的实现细节. 总体而言, 不同启发式评估函数实现方法各有优劣. 遗传算法实现较为简单, 测试生成成功率高, 但需要平衡初始种群选取的随机性和质量, 同时存在子代差异性降低的问题. 符号执行可以用更少的测试达到较高的覆盖率, 且能够生成文本输入测试, 但需要更多的计算资源且受到符号执行技术的限制. 一部分启发式工具具备回溯性, 有意识地记录已探索过状态信息, 并在搜索陷入收益停滞时及时地进行回溯.

表 3　不同启发式工具对比

工具	估值函数设计	回溯性	随机性
EXSYST[62]	启发式+建模	无	高
Sapienz[18]	遗传算法+帕累托最优	无	低
Sapienz-div[63]	遗传算法	无	较高
Evodroid[66]	遗传算法	无	较高
GUICat[67]	符号执行	无	较高
TimeMachine[65]	A*算法	有	高

3.3.3　基于模型的方法

基于模型的测试生成器旨在对系统状态抽象建模, 并在简化的抽象模型上实现动作生成策略. 根据建模方式的不同, 该类方法可进一步分为基于静态分析模型的方法和基于动静结合模型的方法.

基于静态分析模型的方法通过对源码进行静态分析预先构建被测应用的测试空间模型, 该模型通常为有向图形式的界面状态转移模型, 并在测试生成开始之前即完成模型的构建过程, 且中途模型通常不发生改变, 因此模型能否准确反映实际的状态转移关系对测试质量有较大影响. 在测试启动时, 测试生成器通过使用深度优先等图搜索算法对状态转换图中的节点进行逐一遍历, 并生成相应的交互动作序列. 通常情况下, 由测试生成器通过对被测应用源码进行控制流分析得到相应的状态模型, 某些特殊情况下, 研究者会选择人工构建相应模型, 如在 Karlsson 等人[69]的工作中, 为了研究模型测试方法在工业应用上的覆盖率, 研究者选择使用手工模型对被测应用进行表征, 从而完全确保模型准确率. 同时, 人工构建的模型经常作为衡量自动构建的模型精度和生成测试质量的评测基准出现.

Azim 等人[20]提出的 A3E 一般被认为是首个基于模型的测试生成方法. A3E 首先通过静态污点分析获得程序的静态活动迁移图. 在迁移图的基础上实施定向搜索或者深度优先搜索, 其生成的测试覆盖率明显优于人工编写

的同体量测试. 该工具的出现展现了状态空间建模方法在测试生成方向的应用潜力.

A3E 出现后不久, 研究者意识到状态迁移模型的精度对模型测试的效果具有非常重要的影响. 若状态模型粒度过细, 则状态数会随着建模的事件序列指数式增长, 导致路径爆炸, 建模过程无法收敛, 占用大量系统资源. 若状态模型粒度过粗, 包含的状态过少, 则不能完整地记录已探索空间的转移关系, 对后续探索过程起到的指导作用不足, 导致测试质量低. 如何在建模过程中选择合适的模型精度, 平衡模型复杂度和模型表现力这一问题成为模型测试相关研究面临的主要挑战.

Baek 等人[70]提出了一种多维度模型状态划分标准, 从包含组件数量, 所在目录结构, 代码层面等多个角度综合分析不同界面是否应该被划分为同一个模型状态, 使模型状态既不至于过多导致建模难以收敛, 也不至于过少导致精度不足.

Lai 等人[71]提出的 GOALEXPLORER 在已有模型测试工具的基础上, 增加了对于组件组合建模和后台事件的支持. Mirzaei 等人[21]则提出了多级图形界面建模测试框架 SIG_Droid, 实现了对图形界面状态转移过程中的屏幕组件层面和源码层面分别建模. 通过综合两个维度的模型记录已探索过的状态空间, 包含更多信息量的同时, 避免了因为模型精度的提升导致的状态数指数式爆炸问题. Guo 等人[72]在对应用建模的过程中加入依赖关系, 并使用深度优先策略探索模型. 显著提升了生成测试的覆盖率.

Nguyen 等人[73]提出的模型测试框架 GUITAR 将状态建模过程模块化为待测应用剖解、转换图生成、测试脚本生成、测试启动这 4 个阶段, 并支持通过在每个模块内部插入特定方法, 产生基于不同平台的模型测试工具. 后续 Amalfitano 等人[74]又将该测试框架迁移到了移动端, 实现了 MobileGUITAR. MobileGUITAR 通过广度优先搜索探索待测应用中的图形界面状态, 并根据相似度规则合并同类状态, 避免状态爆炸问题.

2017 年, Su 等人[75]提出了一种新的基于模型的图形用户界面测试生成方法 Stoat. Stoat 相较于已有的其他模型测试工具, 主要提出了两个新方法: 其一是采用概率模型描述状态转移关系, 保障精度的同时兼顾解决了状态空间爆炸的问题; 其二是增加了对于测试过程中系统级事件生成的支持. Stoat 集中了已有模型测试研究的多数优势, 是该方向具有代表性的测试生成方法.

基于动静结合模型的方法通过输入探索动作后, 被测应用的反馈结果不断实时修正状态模型. 一旦实际反馈结果与模型预期不一致, 则增量式地将该部分转移关系加入模型, 从而使测试再次进行到同一状态时, 模型可以做出准确预测. 该类建模方法通常较为关注探索动作选择策略的制定.

Gu 等人[22]提出的 APE 实现了在状态建模过程中, 根据探索到的信息量实时调整模型精度. 通过监控测试时的运行时信息, APE 能够动态调整抽象粒度, 使模型具备足够精度的同时不至于过于复杂. Wu 等人[76]在 iOS 端实现了动态检索运行时的 UI 布局及界面组件活跃信息, 并将其用于屏幕状态搜索过程.

Cao 等人[77]提出将人工赋予探索动作优先级与模型测试结合, 利用人工设计的选择算法指导模型测试对未知状态的探索过程: 首先对每一个探索动作赋予相同的优先级, 根据执行后能否到达新状态更新优先级; 每一次选择探索动作, 采用 epslon-软性策略: 以较大的可能选择高优先级的动作, 同时以一定的可能选择其他动作. 其测试覆盖率上的表现优于采用深度优先探索算法的模型测试工具.

Zhang 等人[78]通过结合智能搜索, 状态 fuzz 和间接启动策略等多种技术, 实现了动态导出屏幕状态模型, 达到了比以往方法都要好的模型精度和状态覆盖率.

除上述工作外, 少数测试生成器使用了非界面状态转移模型. 如 Yu 等人[79]使用知识图谱模型表示领域特定知识引导状态空间探索过程, Zhang 等人[80]通过基于马尔可夫链的概率模型实现了测试报告的复现. 成浩亮等人[81]基于手工绘制的图形元素对测试意图进行建模. Sun 等人[82]通过将移动应用中隐私功能相关的时间逻辑序列建模为线性时间逻辑公式 (LTL) 的方式实现了用户隐私相关的界面组件测试用例生成. 在现有工作中, 界面状态转移模型仍是当前基于模型的方法的主要研究方向.

基于模型的测试生成器被广泛应用于除挖掘系统崩溃缺陷之外的其他各类测试任务.

Peng 等人[83]提出的 CAT 通过建模方法对 Android 平台下频繁出现的软件更新内容自动测试. 针对更新内容的测试很难依靠现有的自动化工具实现, 因为它们无法分辨程序原有功能和更新容. CAT 在进行事件搜索之前,

首先将对应用源码中的功能函数和图形界面组件进行逐一映射, 并通过搜索源代码查找隶属于更新内容的功能函数. 通过之前建立的映射关系可以将这些函数映射为界面组件的子集. 接着使用基于模型的测试工具 DroidBot 在当前子集上实施搜索并生成测试. 类似的工作还有 Ngo 等人[84]提出的 ATUA. 该方法采用静态和动态程序分析技术结合的方式, 自动生成扩展的屏幕转移图, 并使用其分辨已有功能和更新的功能.

Ghorbani 等人[85]提出的 DeltaDroid 通过模型测试方法对软件动态交付内容进行测试. 动态交付常见的问题是用户在下载扩展包时时常遭遇网络问题等原因导致的下载失败. DeltaDroid 首先从已有测试集中通过静态和动态分析混合技术得到描述待测应用执行扩展包下载过程的元数据. 之后通过将元数据与扩展包下载失败的原因构建的错误模型相结合, 产生针对某种特定成因的图形用户界面事件序列, 并构建扩展测试.

Fazzini 等人[86]提出的 DiffDroid 采用用户界面状态模型对同一应用不同版本的行为一致性进行测试. 首先利用静态分析工具导出待测应用的用户界面布局, 并将其整合为树的结构, 随后 DiffDroid 会截取屏幕, 并根据匹配算法将截取到的屏幕附加到用户界面结构树的相应节点位置.

Ge 等人[87]将模型测试方法用于辅助众包测试的进行. 众包测试作为一种新型的测试方法, 当前面临的主要问题是测试者缺乏目标应用的使用经验导致写出低质量的测试, 例如重复和误报等测试. 该方法首先经过静态和动态程序分析生成屏幕迁移图, 并根据该图实现测试导出、测试推送、测试引导. 辅助工具将根据测试者当前所在屏幕生成合适的交互动作序列, 并推送给测试者. 测试者在执行这些动作的同时自主判断应用是否出现故障, 从而结合自动测试在覆盖率上的优势和众包测试在测试预言上的优势.

Li 等人针对模型测试方法与人群测试相结合这一领域进行了大量相关研究. 他们提出首先基于静态分析方法得到状态转移图, 并基于深度优先策略遍历转移图得到测试任务列表. 通过这些任务推送给测试参与者, 并基于用户反馈调整任务优先级达到提升综合测试效率及效果的目的[88], 并尝试首先使用模型测试工具进行“预测试”, 并通过静态分析工具导出并收集未覆盖到的界面状态及其路径信息, 测试者只需根据提供的路径信息知道完成对这些未覆盖状态的测试即可[89]. 上述方法均在实验验证中取得了较好的测试效果.

Zhang 等人[80]提出通过状态转移模型引导测试用例复现过程. 测试复现面临的一大难点是报告中描述的事件与实际界面转移动作通常无法精确匹配, 通过引入状态转移模型, 可以实现在全部候选匹配状态中进行图搜索并选取最佳匹配动作, 从而解决丢失或非正确的测试动作匹配问题. 同样关注测试报告的还有 Fang 等人[90]的工作. 该研究通过人工构建的分类模型实现了对测试报告的自动化去重和基于其产生原因的分类整理等工作.

张兴等研究者关注系统空转问题[91]. 在图形用户界面程序中, 系统在处理完业务后并不会退出, 而是界面上空转. 这种空转会影响测试效率. 通过使用基于 Bi-Gram 模型的动态空转检测方法, 能够使进入空转状态的程序及时退出.

Zhao 等人[92]提出的 LIT 通过从实际用户操作中导出上下文相关的策略模型用于指导移动端游戏应用的测试生成, 其效果好于常规测试生成方法.

表 4 展示了本文收录的面向特殊测试任务的模型测试方法及其对应的应用场景.

表 4　面向不同测试任务的模型测试生成方法

相关工作	模型类别	测试任务
Zhang等人[80]	概率模型	面向测试复现
CAT[83]	基于动态分析的状态转移模型	面向更新内容
ATUA[84]	基于动静态结合的扩展屏幕转移模型	面向更新内容
DeltaDroid[85]	基于动静态结合的状态转移模型	面向动态交付程序检测
DiffDroid[86]	基于静态分析的树状界面中转移模型	面向一致性检测
Ge等人[87]	基于静态分析的屏幕转移模型 (AWTG)	面向辅助人群测试
Fang等人[90]	人工模型	面向测试报告分类整理
张兴等人[91]	N-Gram	系统空转检测

值得注意的是, 相较于其他类型的测试生成方法, 基于模型的测试方法在工业界已经有了较为成熟的应用. 例如, Moreira 等人[93]提出的 PBGT Tool, 提供了一个完全集成的测试环境, 提供了建模、配置、自动测试用例生成、自动测试用例执行和测试覆盖率分析的功能. 该工具已被用于多个项目和行业, 并展现了较高的测试生成质量. Ramler 等人[94]介绍了在 3 个不同公司的行业项目中引入模型测试用于自动化图形用户界面测试的例子. 在所有这 3 种情况下, 基于模型的方法均帮助揭示了现有测试无法发现的新缺陷. Wang 等人[95]提出的 VET 在工业级应用上采用基于有向图的搜索算法发现导致搜索陷入闭环的危险状态 (如退出登录), 有效提升了测试效率.

基于模型的测试方法的显著特点在于其建模过程涉及大量计算和数据存储. 因此, 该类方法的算法实现较为复杂且需要较高的计算资源. 然而, 模型测试工具通过模型精确地记录已知的屏幕状态和探索动作的预期收益, 普遍能达到较高的覆盖率. 随着相关技术发展, 模型测试工具逐渐呈现专门化趋势, 研究者不再对完整的屏幕状态空间进行全面建模, 而是选择性地针对特定状态和组件建模, 简化了模型结构并弱化其他状态转移细节.

3.3.4 基于机器学习的方法

机器学习相关技术在移动测试生成领域中发挥着重要作用. 在测试生成过程中, 机器学习方法被广泛应用于系统抽象建模和动作生成策略上. 根据已有针对机器学习技术的分类框架, 可将相关工作分为基于有监督学习的方法, 基于强化学习的方法和基于预训练大语言模型的方法.

基于强化学习的测试生成器通过拟合价值函数评估探索动作的价值, 并预测哪些操作在给定状态中可能导致更好的测试覆盖和更多的缺陷检测数. 该类方法的特点是优化测试生成的效率和效力, 使测试更接近真实用户操作同时保留探索稀缺路径的能力.

特别地, 该类工作经常使用 Q-Learning 等典型的强化学习算法. Q-Learning 是一种逐步学习算法, 首先对所有状态及其转移动作赋予相同的收益, 随着探索的进行, 通过奖惩机制逐步感知并更新收益值, 直到其达到收敛. 对于移动测试生成任务, 可根据是否触发程序崩溃或者达到了新状态进行奖励.

Koroglu 等人[96]提出的 QBE 将一个完整的图形界面事件拆分为响应状态和响应动作, 在进行搜索时, 首先对所有元素赋予相等的执行概率. 随着搜索的深入, 使用激励函数调整执行频率. 一旦执行完某个操作后提高了覆盖率或发现了系统崩溃错误, 激励函数将给该操作对应的矩阵元素赋予更高的执行频率. Adamo 等人[97]同样采用了基于 Q-Learning 的测试生成算法来系统地选择事件并探索被测应用程序的图形用户界面, 避免了图形界面状态模型搭建过程.

传统的 Q-Learning 算法在应用于图形界面测试任务时存在一些不足之处. Q-Learning 给予的奖励值是固定的, 这表示每进行一次更新, 动作收益表会有较大的变化, 可能会导致搜索过程出现不必要的往复跳转. 针对基于 Q-Learning 算法的测试生成过程的优化研究也被广泛关注.

Pan 等人[26]分析了之前基于 Q-Learning 的相关测试生成工具, 并提出了 Q-Learning 的改进方法 Q-testing. Q-testing 采用“好奇驱动”的方法发挥强化学习的作用. 通过缓存一部分访问过的状态, 并将新状态与缓存中的状态比较来设置奖励. 与传统 Q-Learning 相比, 该方法的显著优势在于通过记录已探索状态, 能够避免对差异得分较高但已探索过的状态转移频繁给出较高的 reward, 导致搜索陷入停滞. 实验结果表明该方法优于 Monkey、Stoat[75]和 Sapienz[18].

在强化学习中, 先验知识和历史测试信息对于最终效果有显著影响. 例如, 对于一个需要拖动到底部再点击按钮的信息框, 在没有先验知识的情况下很难生成该事件. 有研究者尝试通过收集其他移动应用的知识来当作模拟的先验知识. 但是该方法会产生过拟合问题, 即只能在与训练集相似的移动应用上表现良好, 缺乏在其他类似的应用上的泛化能力. 历史测试信息同样对测试生成具有重要的指导作用, 例如执行某探索动作可以引发某个状态转移过程等. 充分结合先验知识和历史信息能够为强化学习的决策过程提供更多支持.

2019 年, Degott 等人[98]提出了一种先验知识在强化学习测试生成领域的方法. 将测试生成问题转化为多臂赌博机问题, 使用强化学习来使模型有适应能力, 能够使测试生成器在可能采用的动作无效时, 逐渐切换到不太可能采用的动作. 这是首次将多臂赌博机用于解决测试生成, 并且也是首个将强化学习技术用于测试生成并使模型有适应能力的工作.

Lv 等人[99]提出了将历史测试执行信息指导应用于强化学习决策过程的测试生成工具 Fastbot2. 为了利用历史信息, 他们提出了概率模型以存储该知识, 并在基于模型的方法上使用强化学习方法指导测试的策略. 字节跳动公司近两年的持续记录中, 50.8% 的被修复缺陷是由 Fastbot2 报告.

结合强化学习与其他决策技术的相关研究同样取得了较多的研究成果. 强化学习作为一个独立于任务的决策方法, 可以嵌入测试生成过程的任意阶段, 天然地能与其他测试生成方法相结合.

Koroglu 等人[100]提出了基于时序逻辑公式和强化学习搜索的测试生成工具 FARLEAD-Android. 该工具接收以一种具有较高可读性的形式化语言来描述的测试应用场景, 并将其转换为时序逻辑公式, 根据得到的公式构建激励函数, 使用强化学习的方式引导搜索工具得出可以实现目标应用场景的测试.

Yu 等人[101]利用计算机视觉技术从界面中提取组件布局信息, 并使用 Q-Network 在该信息的指导下选择测试动作, 实现了基于强化学习的跨平台测试生成方法.

YazdaniBanafsheDaragh 等人[102]提出的 Deep GUI 方法利用深度强化学习技术获得应用界面图片的“热力图”, 并使用热力图指导事件的生成, 提升有意义动作的生成概率. Deep GUI 是首次尝试利用跨应用的数据进行训练和测试生成的工作. 同样基于深度强化学习的方法还有 Lan 等人[103]提出的 DQT. 该方法通过图像嵌入技术保存历史测试中的组件布局和语义信息, 并基于 Deep-Q 实现好奇度制导的状态空间探索. Zhao 等人[104]提出的 Dinodroid 也是基于 Deep-Q 网络实现的交互动作选择. Peng 等人[105]实现了基于深度强化学习模型的应用更新内容测试生成. 张少坤等人[106]提出了一种基于多模态表征的移动应用 GUI 模糊测试框架, 通过考虑多模态特征, 如视觉特征、布局上下特征和细粒度的元属性特征来联合推断 GUI 小部件的语义; 然后, 训练一个多层次奖励驱动的深度强化学习模型来优化 GUI 事件选择策略, 提高模糊测试的效率.

Ran 等人[107]提出了一种将强化学习算法和界面建模结合的探索动作选择策略. 首先从交互事件中构建界面布局模型, 并采用基于蒙特卡洛树搜索的强化学习算法对界面事件进行评分, 根据得到的评分对事件探索优先级进行排序. 在生成测试时, 首先参考界面布局模型确认当前界面所在位置, 再根据计算得到的预估回报值由高到低依次尝试该界面下可以生成的探索事件.

基于有监督学习的方法通常利用深度神经网络模型对历史测试信息进行特征提取, 并据此产生新的测试.

Liu 等人[108]在 Monkey 的基础上增加了基于 RNN 模型的分析模块, 用于产生文本类输入内容. 首先提取已有的用户数据中, 用户在特定界面文本框组件下输入的文本以构建测试集, 通过 RNN 学习输入文本与待输入的文本框组件的相关联系, 使得训练后的模型能够根据待输入文本框组件预测所需的输入文本, 同时, 通过语义相似度评估模型 Word2Vec 解决不同应用中使用同义词为相同功能的图形界面组件命名的问题.

White 等人[109]通过训练好的目标识别模型自动识别屏幕上组件类型和位置, 并将模型获得的信息提供给生成器指导测试生成. 从而不需经过 API 调用, 即可直接获取状态信息.

2023 年, Feng 等人[110]实现了一个基于 CNN 模型的测试辅助工具 AdaT. 等待 UI 渲染对于界面测试生成过程十分重要. 等待时间过长会降低测试效率, 等待时间过短导致 UI 渲染过程不完整, 影响测试效果. AdaT 训练了一个基于 CNN 的图像分类模型判定界面渲染进度. 测试启动时, AdaT 不断对应用界面进行截屏, 并通过神经网络模型判断其是否渲染完成. 当判定为渲染完成时, AdaT 通过利用 OpenSTF 实现的测试框架向测试工具发送同步信号以生成下一个事件.

Hu 等人[27]提出的通用测试脚本编写工具 AppFlow 通过多分类网络实现了测试逻辑设计和组件识别的分离. 通过调研已有的同类型应用, 他们构建了一个多分类数据集, 其中通过对组件的文本内容、大小、颜色、位置、OCR 信息等提取特征并形成特征向量, 通过深度神经网络提取这些特征与组件功能的联系, 实现组件自动定位.

基于深度网络模型模拟真实用户操作的测试领域的代表性方法是 Zhao 等人[28]实现的 Avgust. Avgust 通过计算机视觉和自然语言处理的网络模型对包含移动应用真实用户操作的视频中的关键图像和文本进行特征提取, 并据此构建独立于移动应用的上层状态模型. 根据所得状态模型, Avgust 在任何其他应用中产生类似真实用户操作的测试.

Li 等人[111]提出的 Humanoid 同样能够实现模拟人类行为的测试生成. Humanoid 的核心是深度神经网络模型,

它预测人类用户更可能与哪些图形界面元素交互, 以及如何与之交互. 模型的输入是当前图形界面状态和最近的图形界面迁移, 输出是该图形界面状态上每个可能的动作的概率.

Nayak 等人[112]提出利用基于 RNN 的段推荐算法实现模拟真实用户的探索动作选择. 该推荐算法已经被广泛应用于 YouTube 等流媒体, 并取得了较好的效果. 他们将一段时间或一次服务流程内, 用户与界面的交互动作加工成段, 并通过 RNN 模型从这些历史交互信息中学习用户行为模式.

Hu 等人[113]提出的 Appaction 基于多模态模型实现了模拟真实用户动作输入的测试生成策略, 该方法已实际应用于美团等商业软件的测试过程, 并取得较好的效果.

Saha 等人[114]调研了机器学习方法在定位存在缺陷的 GUI 界面和 GUI 组件方面的能力. 他们对比了一种基于文本的模型, 两种多模态模型和一种无监督学习方法, 发现处理图像特征的模型更擅长定位缺陷 GUI 界面, 处理文本特征的模型则更擅长定位缺陷的 GUI 组件.

Liu 等人[115]提出的 APICOG 结合机器学习和自然语言处理技术实现了涉及用户隐私的交互动作安全性测试. 该方法首先使用 NLP 技术导出涉及隐私的 API 调用过程所在的 GUI 状态及上下文信息, 通过深度神经网络分析这些信息并预测该处 API 调用是否必要且合法.

表 5 整理了本文收录的基于有监督学习的测试生成方法及其对应的测试任务分布情况.

表 5　基于有监督学习的测试生成方法

相关工作	模型类别	测试任务
Liu等人[108]	RNN	生成文本输入
White等人[109]	CNN	生成探索动作
AdaT[110]	CNN	提升测试执行效率
AppFlow[27]	多分类FCN	生成探索动作
Avgust[28]	RNN	模拟真实用户输入
Humanoid[111]	RNN	模拟真实用户输入
Nayak等人[112]	RNN	模拟真实用户输入
Appaction[113]	多模态深度学习模型	生成探索动作

基于深度学习的测试生成方法在工业应用上的测试表现同样面临一些挑战. Peng 等人[116]在使用 Humanoid 对他们的软件产品进行测试时, 就发现其存在两处缺点, 即界面结构表示不够简洁和单模态预测模型表现力不足, 导致其不适用于商业化应用.

自从 2022 年底以来, 随着 ChatGPT 的流行, 预训练大语言模型技术吸引了广泛关注. 在移动测试生成过程中, 大模型既可以像传统决策模型一样, 通过学习用户数据指导探索动作选择, 也能通过发挥其擅长的文字处理能力, 将自然语言描述的测试过程转换为具体的测试步骤. 二者需要重点关注的是设计有效提示词 (prompt). 大模型当前能够记忆的信息条数有限, 如何将界面状态信息和测试任务通过编写合适的提示词传递给模型是十分关键的. 现有主流方法是使用提示学习的方式生成提示词.

大语言模型的自然语言理解能力使其能够实现将自然语言描述的测试过程转换为实际的测试动作集合的过程. Zimmermann 等人[117]提出了一种利用 GPT-3 实现应用界面自动测试的方法. 为了克服 GPT-3 对输入条数记忆能力有限的缺点, 他们提出将每一次交互后用户界面的全部状态信息整合为一个提示词来输入. Taeb 等人[118]通过结合大语言模型和基于像素的用户界面理解模型实现了将自然语言描述的可用性测试流程转换为测试脚本的过程.

Liu 等人[29]提出的 GPTDroid 通过传入界面信息要求大模型生成相应测试脚本, 执行脚本的同时不断将状态变化反馈给大模型, 从而生成连续的测试动作序列.

Yoon 等人[119]进一步提出测试者不直接给出具体测试过程描述, 仅给出测试要求, 测试生成器通过大模型自动创建测试目标并完成测试生成过程. 该方法同样取得了较好的效果.

Wang 等人[120]提出将大模型应用于测试脚本复现, 并实现了基于 GPT-4 的反馈制导的测试复现工具 ReBL.

不同于传统测试复现方法的逐步骤复现模式, 该方法直接基于整个报告文本设计提示词向大模型提供必要的上下文推理信息. 实验证明, 该方法除大幅提高复现精度和效率外, 还具备复现功能缺陷测试脚本的能力.

对于文本输入生成领域而言, 大模型执行的任务与深度神经网络模型类似, 即通过已有文本输入学习相应特征, 据此生成新的输入. 通过采用小样本学习或上下文学习等手段, 大模型能够产生比传统机器学习方法质量更高的文本输入.

Liu 等人[30]结合大规模预训练模型技术提出了基于情景感知的文本测试自动生成方法 QTypist. 该方法能够自动从应用的界面布局设计文件中导出上下文信息, 并自动生成输入大模型的提示词. 为了加速这一过程, 他们还开发了一套自动从应用布局中导出组件信息, 从而自动构建提示词及其答案格式的数据库, 用于进行提示学习. Liu 等人[121]后续又提出了通过大模型指导生成代码变异算子的方式产生独特的文本输入的方法.

随着机器学习技术的快速发展, 相关研究取得了显著进展. 深度学习在图像和文本等数据类型的分析和理解方面表现出强大能力, 使得这一方法能够模拟人类测试员生成并选择测试动作. 特别是随着预训练大语言模型等新技术的出现, 机器学习技术在理解移动应用中的图像、代码和文本描述等多模态数据方面的效果得到了进一步提升, 从而能够更好地理解动作语义.

3.3.5 基于测试迁移的方法

测试迁移是一类特殊的测试自动生成方法, 其探索动作的生成方式和其他方法不同. 该类方法的测试环境提供的模型 M 中样例输入集合 I 不为空. 并且该类方法不关注探索动作是否对发现新状态有贡献, 而是将已有源应用上的探索动作转化为目标应用上的探索动作. 测试迁移实现的是已有测试逻辑的自动复用, 而非从零生成探索动作. 因此测试迁移可以在崩溃等被动预言之外主动生成预言. 通过对已有测试预言进行转化和迁移即可生成目标应用上的测试预言. 上述特点也决定了测试迁移是一类应用范围相对狭窄的测试生成方法. 仅当源应用程序存在且包含大量已有测试脚本时, 测试迁移方法才可取得较好的效果.

测试迁移的主要应用场景可以被总结为以下 4 个类别.

(1) 同一个应用的跨操作系统迁移, 如同一应用的 Android 端到 iOS 端的测试脚本迁移.

(2) 同一个应用的跨架构迁移, 如同一应用的 Web 端到移动应用端的测试脚本复用.

(3) 同一个应用的跨设备迁移, 如同一应用的 PC 客户端到 Android 客户端之间的脚本迁移.

(4) 同类型应用的同设备迁移, 如支付工具的测试脚本迁移.

上述各应用场景的迁移操作难度逐渐递增. Andorid 和 iOS 之间的迁移只要将事件序列中的节点一一映射即可. 跨架构迁移需要考虑 Web 图形界面组件捕捉任务, 因为 Web 端的绝大多数图形界面组件的定义隐藏在嵌入式的 JavaScript 代码中, 使用单纯的 HTML 解析工具不能定位到全部组件. 跨设备迁移除了要进行事件序列匹配, 还要考虑设备支持的操作集合的匹配, 例如移动端支持各种手势而 PC 端则支持的是键盘鼠标输入. 同类型应用之间的测试迁移仅能复用一部分测试预言相关的事件序列, 其他的事件需要自行探索补全. 在实践中, 最常见的情况是某企业首先推出了一个移动应用并进行了相关测试, 运行了一段时间后又推出了不同操作系统下的版本. 此时, 企业将已有相同或者相似应用的测试脚本的进行复用, 从而节省测试成本.

由于前 3 个类别涉及的技术具备一定相似性, 本节将从基于测试动作匹配的同应用跨平台迁移及基于语义提取的同类型应用迁移两个方向对相关工作展开介绍.

对于基于测试动作匹配的迁移方法而言, 令待迁移应用为 S , 目标应用为 T , 该类生成工具可获得的 S 和 T 的状态全集, 并在给定输入的情况下尝试尽可能地将 S 中的状态映射到 T 中的状态. 根据状态匹配发生的时机, 我们可以将事件匹配算法划分为静态匹配算法、动态匹配算法、动静态结合的匹配算法.

动态匹配基于实时反馈进行事件匹配: 考虑用 $event_i = (w_i, a_i)$ 来描述一个图形界面事件, 其中 w 表示触发该事件的图形界面组件, a 表示对该组件进行的操作. 动态匹配首先启动 T , 并记录已经匹配的 S 的测试用例中的事件序列长度 i , 并遍历当前 T 的界面中的所有组件, 找出与当前待匹配事件 $event_{i+1}+1$ 中的 w_i+1 相似度最高的组件, 并触发事件 a_i+1 . 此时 T 将进入一个新的界面. 该方法重复上述工作, 直至 S 的测试用例中的所有事件都被成功匹配. 动态匹配面临的挑战是 T 中的每一次状态变化都必须严格匹配 S 中的一个事件, 否则算法无法推进. 但是在

大多数情况下, S 和 T 中的图形界面事件并不能完美一一对应.

静态匹配算法首先通过静态分析工具导出目标应用 T 的状态空间对应的抽象模型, 将 S 的测试脚本中的事件与模型状态匹配, 并根据模型的状态转移关系自动补全没有对应关系的状态转移事件. 静态匹配不要求逐一匹配, 但匹配精度依赖模型精度.

多数实际相关工作采用的则是动静结合的组件匹配策略: 首先执行动态匹配, 出现当前界面上的任何组件操作均无法严格匹配待迁移事件序列中的个体时, 采用静态建模的程序迁移图对探索动作进行指导. 常用的方法是通过 k-近邻算法扩展事件匹配样本空间, 探索当前界面经过 k 步动作后可达的界面中的组件操作, 并将其纳入待匹配事件候选空间. 动静结合的匹配方法同样遵循顺序匹配原则, 需要确保待迁移事件序列和目标应用 T 中的组件动作最终的发生前后顺序一致.

在极少数情况下, 动静结合的匹配策略无法替代单独使用静态或动态匹配的方法. 例如, 在待迁移应用存在一定恶意软件风险时, 需要尽可能避免实际执行探索动作. 此时, 即使这将丢失一部分匹配精度, 迁移过程也更偏向使用基于纯静态分析得到的状态转移结果进行.

Behrang 等人[122]提出的 GUITestMigrator 是移动端的测试迁移的早期代表性工作. 该工具首先分析源应用的图形用户界面事件并提取测试场景集合, 之后在目标应用中尝试匹配场景, 最后对匹配到的场景迁移测试. 尽管迁移的对象应用是代码量较小, 结构简单且功能相似的应用, 但其迁移成功率展示了移动端测试脚本复用的可行性. Behrang 等人[23]后续在 GUITestMigrator 的基础上进行了多个方面的改进, 提出了新方法 ATM. 尤其是引入了基于静态分析的状态空间剪枝策略, 优化了 GUITestMigrator 完全基于动态分析的事件匹配算法.

Qin 等人[24]提出的 TestMig 通过递推公式解决复杂的多对多事件匹配问题. 该方法在单个事件相似度评估方式已知的情况下, 对 S 中的 k 个事件和 T 中的 L 个事件的匹配过程可以递归地划分为对 S 中的前 $k-1$ 个事件和 T 中的一部分事件进行匹配和对 S 中的第 K 个事件与 T 中剩余事件进行匹配两个步骤.

TestMig 可以处理 Android 和 iOS 之间已有测试脚本的互相迁移, 然而有些情况下, 不同平台下的应用都不存在足量的人工测试脚本. 此时从用户操作历史中导出测试逻辑并实现迁移是一个不错的解决方法. Yu 等人[123]提出的 LIRAT 能够从用户操作截屏中导出跨平台测试脚本. LIRAT 首先通过计算机视觉技术从包含测试需求的操作截屏中导出复现该操作所需的组件布局信息等视觉特征, 并综合使用组件特征匹配和布局特征匹配策略将组件与提取到的特征进行匹配, 并产生面向特定平台的测试脚本. LIRAT 同时支持产生面向 Android 和 iOS 平台的测试脚本. LIRAT 应用了用户反馈数据, 在基于 Android 和 iOS 平台的测试迁移研究领域提供了一个新的颇具价值的研究思路.

除了人工编写匹配规则以外, 部分相关工作通过引入其他技术实现了自动匹配. 例如, Zhang 等人[124]提出使用 BERT 学习上下文语义信息的方式实现组件自动匹配. Ji 等人[125]探讨了基于组件视觉特征的动作事件匹配方法实现的可能性. 他们指出, 目前待测应用中相当一部分待匹配的组件对存在较大的外观差异, 使得基于外观相似度匹配变得非常困难, 当前基于视觉特征识别的方法仍存在较大的改进空间.

跨设备和架构的测试迁移的方法中的代表性工作是 Lin 等人[126]提出的 TransDroid. 该方法通过将 Web 端图形界面事件采用 4 种预处理方式: 可以直接匹配的事件不做处理, 具有逻辑关系的事件进行融合, Android 端存在类似功能的事件进行变形, Android 端不存在的事件进行删除, 最终解决移动端和 PC 端支持的操作集合不一致问题.

基于语义提取的生成方法主要应用于同类型应用之间的测试复用. 同类型应用的组件命名风格一般存在较大差异, 相似度也更低, 因此在匹配之前需要进行一些预处理. 当前使用较多的一种预处理做法是通过 word2vec 等手段对组件文本进行语义分析, 依据组件功能的语义相似性进行匹配.

Mariani 等人[25]通过实证研究分析了语义提取预处理对组件相似度匹配精确度的重要影响. 该工作从文档语料库、词嵌入技术、事件描述提取器和语义匹配算法 4 个方面讨论了语义匹配对于最终迁移精度的影响, 并提出了新的语义匹配方法 SemFinder 和新语料库. 该工作基于他们之前在功能语义匹配上的研究[41], 即通过提取一些独立于应用的功能实现应用间的测试脚本复用. 所谓“独立于应用的功能”包括身份验证操作、增删改查操作、搜

索和预定操作等移动应用普遍支持的操作.

Lin 等人[127]提出的 CraftDroid 实现了基于信息检索, 将已有移动应用的测试重用在其他应用上. 由于针对相似功能和特征的测试, 该方法能够自动继承原有测试的功能和表现力. 且同时支持迁移测试预言.

Mariani 等人[128]系统地总结了同类型应用测试迁移的 3 大难点, 即巨大的屏幕搜索空间、存在较大差异的组件布局、交互事件的非一一对应性. 在此基础上, 他们提出结合了遗传算法的新测试迁移方法 AdaptDroid. 该方法通过遗传算法生成并选择与待迁移脚本语义上最相似的交互序列, 从而迅速缩小搜索空间, 并采用了宽松的组件匹配策略, 仅考虑语义相似性, 不考虑外观相似性. 对于事件非对应问题, 采用灵活的匹配策略, 综合考虑待测应用和被迁移应用的特性. 其迁移正确率优于 CraftDroid.

Zhang 等人[129]提出的测试迁移增强工具 Migrate 能够基于多个针对相同功能的待迁移测试合成新的功能测试. 该方法首先将这些功能测试中涉及的界面状态全部导出并组合得到总体状态序列, 随后在总体序列中移除无效事件和断言, 并调整若干衔接事件以确保剩余测试动作能够在待迁移应用中运行. 该方法成功提高了 3 种主流测试迁移工具, 即 CraftDroid、AppFlow、ATM 的功能覆盖率.

近年来, 该领域兴起了一种新的研究思路, 即不考虑严格匹配测试过程, 仅对测试逻辑进行复用. Jha 等人[130]认为面向相同应用领域 (办公、娱乐、消费等) 的应用具有类似的测试逻辑, 据此, 他们提出了一套测试自动复用框架. 首先搜集大量包含测试的应用, 并切分为单元化的测试片段, 根据专业知识对应用进行功能分类. 实施复用时, 首先对待测应用进行领域分析, 确定其所属功能类别, 并通过对该类别已有的单元测试片段的复用和拼接, 生成针对待测应用的测试

关于测试迁移工具的迁移正确率的评估尚不存在一个统一的标准. Zhao 等人[131]提出了面向测试脚本复用工具的自动化性能评判工具 FrUITer. FrUITer 综合了 7 个相关工具在评估转化精度时采用的计算指标, 并根据调研相关背景知识对其进行加权处理, 得到一个无偏向的最终精确度指标. 同时, FrUITer 还引入了自己定义的削减量指标, 量化地评估采用转化工具转化和人工编写相同的测试相比节省的工作量.

测试迁移技术由于不存在状态空间自动探索动作, 而是直接复用人工编写的测试逻辑, 能够保证测试的质量并极大地缓解了缺乏测试预言的问题. 广义上的测试迁移包括跨各种软硬件运行环境的测试脚本复用. 其中, 基于 Andorid 端和 iOS 端的同款应用的测试脚本复用相关研究占据主导地位. 该研究方向当前重点研究内容是处理复杂的多对多事件匹配关系. 同类型应用的测试脚本复用当前的关注点在于组件文本语义提取方法和语义相似度评估策略. 移动端和 PC 端之间的相同应用测试脚本复用由于实现难度偏高等原因尚无研究.

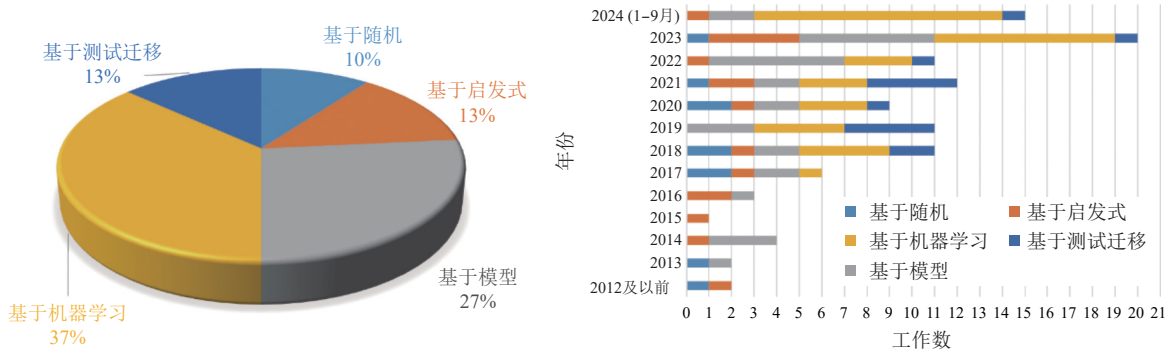
3.4 小　结

本节总结移动测试生成方法的研究现状、发展脉络和各自的特点. 在前文所描述的“测试生成器-测试环境”的研究框架下, 本节内容从测试生成器的视角将现有测试生成方法分为基于随机、基于启发式搜索、基于模型、基于机器学习和基于测试迁移这 5 类方法. 移动测试生成方法以生成高效的探索动作序列为核心任务, 即在有限的动作数量下尽可能覆盖更多的应用界面状态并最终发现更多真实缺陷.

在不同类别的测试生成方法的发展趋势方面, 上述类别中最先出现的是基于随机的方法, 即 Android 官方推出的 Monkey 以及后续研究者在其基础上的改进版. 该类方法原理简单, 部署和应用难度较低, 不依赖复杂的测试环境. 因此, 该类方法至今仍然是学术界和工业界常用的测试生成方法. 随后, 通过在随机算法的基础上添加启发式规则引导, 演化出了基于启发式搜索的方法. 另一方面, 基于模型的方法则早在 2000 年左右就被提出用于测试单机桌面图形界面程序, 因此在相关研究早期, 就有研究者借鉴并迁移基于模型的方法用于生成针对移动应用程序的测试. 在此之后, 随着机器学习技术, 特别是深度学习技术的发展, 在 2018 年之后研究者陆续开始提出基于机器学习的生成方法. 该类方法借助强化学习、图形处理、自然语言处理等技术的发展, 取得了显著的进步. 特别地, 近年来大模型相关技术的兴起对移动测试生成领域产生了广泛而深刻的影响. 在可以预见的将来, 基于机器学习的方法依旧会是该研究方向的热点. 最后, 基于测试迁移方法是近年来被提出的, 是最晚出现的方法. 移动平台同质化和不同行业应用的规范化, 为测试迁移方法提供了实现的可能. 该类方法需要分析、匹配、聚类源应用和

目标应用的状态和组件, 因此也经常使用机器学习方法完成生成任务. 总的来说, 5 类方法经历了探索策略从简单到复杂、所依赖的信息从少到多、所使用的技术从单一到多样的发展演化趋势.

图 3 详细展示了本文收录的基于不同测试生成方法的相关工作的数量分布情况. 图 3(a) 统计了 2012–2024 年 9 月期间, 各类测试生成方法相关工作数量总和的分布情况. 由图可见, 移动应用用户界面自动测试生成主题在科研领域相对更受关注的是基于模型和基于机器学习的方法. 基于随机的方法虽然在工业界有最广泛的应用, 但由于已有工具已经较为稳定, 科研领域的工作相对数量较少. 图 3(b) 按年份详细展示了该时间段内相关工作数量分布逐年变化的趋势, 经分析可以得知, 最先出现的是基于随机和基于启发式的方法. 基于模型的方法自被引入移动应用测试领域后每年保持着相对稳定的研究数量, 基于机器学习的方法诞生较晚, 自 2018 年才开始出现相关工作, 但研究数量存在逐年小幅增加的趋势. 基于随机的方法在中间几年的关注度下降. 随着待布局重构和随机测试与人工测试混合等新思路的出现, 自 2020 年起该方法又开始重新出现相关研究.



(a) (b)

图 3 基于不同测试生成方法的相关工作分布统计

为了横向对比不同方法的整体特点, 本文从使用难度、测试生成的成功率、能否生成测试预言、方法复杂度和是否依赖测试环境输入等多个维度对上述方法进行梳理. 如表 6 所示. 从表中的数据可见, 该领域的研究整体上随着方法复杂度的提升, 使用难度的增加, 所需测试环境提供的信息的增多, 测试生成器逐渐拥有更强的测试预言和系统级事件的生成能力. 所生成的测试是否能成功执行, 与所选用的测试动作集合和所针对的界面复杂程度有关. 因此, 测试生成的成功率是一个重要的参考指标, 但是不能单纯凭借生成成功率对测试生成方法进行评判.

表 6 基于不同探索动作的测试生成工具及其特点

测试生成方法类别	使用难度	测试生成成功率	测试预言生成	系统级事件生成	复杂度	测试环境输入
基于随机	低	非常高	无	部分有	低	无
基于启发式搜索	较低	高	无	多数有	较高	启发式信息
基于模型	高	较低	少数有	无	高	无
基于机器学习	较高	高	基本无	多数有	较高	用户数据等
基于测试迁移	较低	较高	有	无	高	已有测试脚本

4 重要测试数据集和开源工具总结

本节总结前文描述的方法中较有影响的基准实验数据集和重要测试生成工具以便读者检索.

4.1 重要开源数据集

数据集对于方法的构建和缺陷类型的分析具有重要意义. 研究者调研开源社区的真实缺陷和缺陷报告等开发历史数据, 总结收集了不同特点的数据集. 本文介绍其中较为完善的数据集. 本节收录的数据集由第 3 节列举的测试生成器适配的数据集及 4 篇实证研究类工作中提出的数据集[54,132–134]构成.

值得一提的是, 钟怡等人[135]最新提出并实现的评估框架 CEAT 规范了验证步骤并结合了多种评测指标. Lan 等人[136]最新的实证研究工作同样重点关注了当前主要基于测试覆盖率和缺陷发现数的传统评测指标设计无法完整客观地反映测试工具实际性能的问题, 并具体分析了常用评测指标之间的相关性, 及不同指标在不同维度上反映测试生成工具性能的优势和劣势. 上述研究工作有效推动了现有测试生成方法验证过程实验设计的标准化.

表 7 中列举了该领域影响较大的数据集, 涵盖了测试生成、组件定位、真实移动应用缺陷、框架相关缺陷、显示缺陷和面向残障人士可用性缺陷等多个不同方面. 表 7 中总共收集了 21 个数据集, 并按照其出现顺序排列. 我们可以看到相关数据集也存在明显的演化趋势, 即通用数据集的规模不断增大, 专用数据集不断涌现. 在早期数据集包含的缺陷数和移动应用数普遍较少, 随后较大规模的开源数据集开始出现. 随着一些诸如面向残障人士应用测试等专门领域的相关研究增多, 具有特定用途和特定缺陷的数据集也逐渐涌现.

表 7　重要数据集一览

提出年份	参考文献	分类	测试工具分类	简介
2017	[64]	真实用户操作	基于启发式	包含434个参与者的人群测试结果
2017	[108]	真实用户操作	基于机器学习	训练用数据集, 真实用户操作序列组成的向量
2018	[37]	缺陷报告集合	基于模型	由作者的测试工具发现的新缺陷集合
2018	[27]	应用截屏	基于机器学习	人工构建的带标签数据集, 包含组件视觉特征及其描述文本
2018	[41]	测试集合	基于测试迁移	21个完整或部分测试脚本, 用于提取应用无关的功能
2019	[98]	源码集合	基于模型	作者从开源应用和商业应用中挑选并构造的小规模测试用应用集合
2019	[109]	应用截屏	基于机器学习	作者使用Java Swing随机生成的测试用图形界面
2019	[23]	源码集合	基于测试迁移	4类总计16个相同功能的Android应用
2020	[112]	真实用户操作	基于机器学习	作者用于训练推荐模型用的数据集
2020	[54]	缺陷报告集合	实证研究	作者从大量开源和商业应用中收集到的纯系统崩溃缺陷分析数据集
2021	[83]	测试集合	基于模型	测试工具在21个真实应用上生成的测试脚本集合
2021	[25]	源码集合	基于测试迁移	从ATM和CraftDroid的测试集合中挑选的应用
2022	[39]	缺陷报告集合	基于启发式	作者从App Store和推特上搜集的867个热门应用的用户反馈缺陷
2022	[76]	缺陷报告集合	基于模型	测试工具在57个真实应用上发现的新缺陷集合
2022	[84]	源码集合	基于模型	从25个应用中收集的51个更新包源码
2022	[28]	应用截屏	基于机器学习	从20个真实应用中导出的374段用户操作录像
2023	[90]	真实用户操作	基于模型	测试辅助工具在20个开源应用上的运行结果
2023	[107]	源码集合	基于机器学习	从前人总结的数据集中挑选了21个当前使用率较高的应用
2023	[132]	应用截屏	实证研究	从30个真实应用中导出的组件布局特征
2024	[133]	源码集合	实证研究	不同测试生成器在11个Android应用上的效果对比
2024	[134]	应用截屏	实证研究	62个开源应用中导出的116个用户界面及相关上下文信息

4.2 重要测试工具

本节整理了基于不同探索方式的移动测试生成领域的代表性相关工具. 作为对第 3 节提出的分类框架的补充, 我们在本节将从测试生成器面临的测试任务的维度展开, 对前述章节中介绍的相关工作进行整理和展示. 通过对已有工作的整理, 我们提出从测试生成器的应用对象和测试目标两个维度对测试任务进行考察. 对于大部分测试生成器, 其应用对象是整个被测应用, 最终测试目标是发现能够影响正常使用的用户界面缺陷. 除此之外, 测试生成器面临的测试任务主要还包括如下几种.

(1) 易用性测试 (accessibility testing): 应用对象是整个被测应用, 测试目标是评估被测应用界面设计是否方便使用和理解, 挖掘可能增加使用难度的设计模式.

(2) 针对更新内容的测试 (change-focused testing): 应用对象局限于被测应用的特定更新内容, 测试目标和常规用户界面测试一致.

(3) 用户界面组件定位 (widget location): 应用对象是整个被测应用, 测试目标是对被测应用中的 GUI 组件进行识别和定位, 为后续生成交互动作做准备.

(4) 测试辅助: 应用对象是现有测试方法, 测试目标是提升其测试效率和测试质量.

表 8 针对不同测试任务按时间顺序整理了相应的测试生成工具, 表中共收录了 11 个相关工作, 并依次列出了工具名称, 发表时间, 测试任务和所用的技术.

表 8 特殊测试任务的重要测试生成工具一览

名称	参考文献	发表年份	测试任务	所用技术
MATE	[37]	2018	易用性测试	启发式搜索
Groundhog	[12]	2021	易用性测试	启发式搜索
AccessiText	[39]	2022	易用性测试	动态程序分析
AXNav	[118]	2024	易用性测试	大语言模型
CAT	[83]	2021	更新内容测试	状态转移模型
ATUA	[84]	2022	更新内容测试	状态转移模型
Hawkeye	[105]	2024	更新内容测试	深度强化学习
AppFlow	[27]	2018	组件定位	深度神经网络
TOGGLE	[15]	2021	组件定位	计算机视觉
TOGGLE+	[46]	2021	组件定位	计算机视觉
AdaT	[110]	2023	测试辅助	深度神经网络模型

由于一般测试任务的测试生成器数量较多, 本节只收录开源的测试工具. 表 9 按照时间顺序展示了符合条件的测试工具. 表中共收录了 26 个开源工具. 表 9 中的链接收集于 2024 年 9 月.

表 9 一般测试任务的重要开源测试生成工具一览

名称	参考文献	发表年份	方法分类	链接
Dynodroid	[16]	2013	基于随机	https://github.com/dynodroid/dynodroid
A3E	[20]	2013	基于模型	https://github.com/tanzirul/a3e
Evodroid	[66]	2014	基于启发式	https://github.com/sam2kb/evodroid
Sapienz	[18]	2016	基于启发式	http://github.com/Rhapsod/sapienz
DroidMate	[57]	2016	基于随机	https://github.com/konrad-jamrozik/droidmate
DetReduce	[31]	2018	基于启发式	https://github.com/wtchoi/swifthand2
Augusto	[41]	2018	基于测试迁移	http://github.com/danydunk/Augusto
GOALEXPLORER	[71]	2019	基于模型	https://resess.github.io/PaperAppendices/GoalExplorer/
TestMig	[24]	2019	基于测试迁移	https://sites.google.com/site/testmigicse2019
CraftDroid	[127]	2019	基于测试迁移	https://github.com/seal-hub/CraftDroid
ATM	[23]	2019	基于测试迁移	https://sites.google.com/view/apptestmigrator/
APE	[22]	2019	基于模型	http://gutianxiao.com/ape
TimeMachine	[65]	2020	基于启发式	https://github.com/DroidTest/TimeMachine2
FrUITeR	[131]	2020	基于测试迁移	https://felicitia.github.io/FrUITeR/
Deep GUI	[102]	2021	基于机器学习	https://github.com/Feri73/deep-gui
VET	[95]	2021	基于启发式	https://github.com/VET-UI-Testing/main
Sapienz-div	[63]	2021	基于启发式	https://github.com/thomas-vogel/sapienzdiv-ssbse19
TransDroid	[126]	2022	基于测试迁移	https://sites.google.com/view/icst22-transdroid
iFixDataLoss	[50]	2022	基于模型	https://github.com/iFixDataLoss/iFixDataloss22
Fastbot2	[99]	2022	基于机器学习	https://github.com/bytedance/Fastbot_Android
Avgust	[28]	2022	基于机器学习	https://zenodo.org/record/7036218#.ZFyXWHZBw7E
PSDroid	[48]	2023	基于模型	https://github.com/PSDroid2022
Badge	[107]	2023	基于模型	https://github.com/ranpku/Badge
Columbus	[19]	2023	基于启发式	https://github.com/ucsb-seclab/columbus
CydiOS	[76]	2023	基于模型	https://github.com/SoftWare2022Testing/CydiOS
QTypist	[30]	2023	基于机器学习	https://github.com/franklinbill/QTypist

5 当前挑战和未来研究方向

本节首先分析当前技术面临的挑战, 随后对未来研究展开讨论.

5.1 当前面临的挑战

虽然移动测试生成领域在近 10 年的发展中取得了长足的进步, 但目前相关成果仅有少数在工业界落地, 相关研究距离大规模工业应用仍存在较大距离. Linares-Vásquez 等人[137]整合了不同研究者提出的现有测试生成工具在技术层面的优化方向, 提出了理想测试生成工具应该具备的 3 个特点: 延续性 (能够在不同测试环境/不同测试目的条件下运行测试)、进步性 (能够从已有测试结果中总结规律并不断提高产生测试的质量)、大规模性 (测试规模足够大, 足以模拟大量用户使用目标应用的真实场景). 从这 3 个角度出发, 可以发现该领域仍面临相当大的挑战. 这其中既存在技术层面的, 也存在非技术层面的.

在技术层面, 该领域存在的挑战主要有以下几点.

(1) 测试预言的自动生成

测试预言是所有测试自动生成问题长期存在的挑战. 在移动测试生成领域, 测试预言用于描述在给定组件交互动作输入下目标应用的预期界面状态转移过程. 目前大多数工具仅使用系统崩溃等易获取的系统状态作为被动生成的测试预言, 难以覆盖类型多样, 根因复杂的移动应用界面缺陷. 少数涉及主动生成功能类缺陷预言的工作也难以提供与全部测试动作相匹配的完整测试预言.

(2) 移动平台的碎片化

移动应用涉及各种设备、操作系统、屏幕尺寸和分辨率等因素, 导致测试生成过程需要考虑不同的环境和配置. 此外, 移动应用通常与网络、传感器、位置服务等外部因素交互, 增加了测试生成过程的复杂性. 有研究者指出现有工作缺乏对跨平台的支持[138]. 移动端不稳定的运行环境和有限的内存、处理器 、显示等系统资源限制了测试生成工具的发挥[4]. 自动生成的事件序列缺乏对移动端系统级上下文信息的感知等[6]也制约了生成的测试质量.

(3) 移动应用的交互方式复杂

测试生成需要考虑用户操作、手势、触摸屏、滑动、旋转等各种交互方式, 并生成相应的测试用例来覆盖不同的用户交互路径. 这需要综合考虑多个因素, 包括组件的定位和用户行为模式的建模等. 有众多研究者注意到这个挑战. 有研究者指出现有测试工具复现相同错误比较繁琐[30], 并且测试工具缺乏对回归测试等阶段的支持[137].

(4) 生成揭错能力强的测试

现有研究在生成揭错能力较强的测试上仍然存在不足. Su 等人[139]对于测试生成工具挖掘系统崩溃类错误的能力进行过一项重要的实证研究. 他们收集了 52 个真实系统崩溃类缺陷, 并集成了 Monkey、APE、TimeMachine 和 Q-testing 等最新的工具进行研究. 在总计多达 10 920 小时的 CPU 时间的实验中, 18 个真实缺陷没有被任何一个工具发现. 即便是仅针对崩溃缺陷, 当前测试生成工具仍存在较大的进步空间.

(5) 易于安装和使用

易安装性和易使用性是评价工具的重要指标, 现有自动测试生成工具大多不满足这些要求. 在 Linares-Vásquez 等人[140]的调研中, 比起研究者普遍关注的代码覆盖率等量化评估指标, 软件测试人员的关注角度更加多维, 例如开发者在编写测试用例时十分看重用户数据, 偏向采用可以灵活部署, 能快速跟随用户需求变动而调整的测试. Banerjee 等人[5]曾对 1991–2011 年间的 136 篇图形界面测试相关论文进行了分类整理工作, 发现在当时基于模型的测试是主流研究方向, 但工业测试常用的 Monkey 和 Quick Test Pro 等工具并非基于模型. 造成这一现象的原因是 Monkey 等工具更容易部署.

在非技术层面, 该领域存在的挑战主要有以下两点.

(1) 测试普及程度较低

尽管移动测试生成领域吸引了较多研究者的关注, 大量调研结果显示相关工具在工业化的移动软件测试中的

普及率并不高. 在 2020 年, Lin 等人[141]分析了 12 000 个真实 Android 应用, 发现只有不到 8% 的应用附带有测试. Coppla 等人[142]特定针对自动测试的普及率进行统计, 发现仅有 14% 的应用在其安装文档中提供测试代码, 仅有 9% 的应用提供可执行的自动化测试工具. 这证明该领域研究成果距离工业应用还有一定距离. 值得关注的是, 近年的研究中, 有越来越多的研究者与企业合作, 在论文中报告发现商业应用的真实缺陷. 例如, Sun 等人[143]的研究就报告检测出了微信、抖音等流行的移动应用中的缺陷.

(2) 工具缺乏组织和维护

移动测试生成工具经常出现代码缺乏系统的组织和维护的情况. Said 等人[7]的研究显示, 在该领域中大量的研究工具没有被及时地维护、调试和更新, 导致其无法被大规模应用. 而且, 虽然该领域在软件工程领域有专门的研讨会等学术交流活动, 但缺少一个公认较为权威的平台, 用于展示和收集已有方法、工具和数据集. 这无疑不利于该领域的发展.

5.2 未来研究方向

根据前文提及的挑战, 未来研究除了深入研究挖掘测试预言和提升测试揭错能力等技术性问题, 也应该注重该领域的生态与社区建设. 本文认为, 未来可能从以下几个方面展开研究以促进移动测试生成领域的发展.

(1) 结合大模型等人工智能新技术探索新的测试生成方法

测试生成器的生成策略是该领域的核心. 高效地生成有效的测试动作是该领域的核心任务. 根据第 3 节中针对不同生成方法的演化趋势的相关分析, 我们可以发现测试生成技术越来越多地与人工智能领域新方法和新技术结合. 尤其是近来兴起的预训练大语言模型相关技术, 在自然语言处理、程序理解和生成、多模态等任务上表现出巨大的优势. 研究者可以借助相关领域的最新成果提出新的测试生成方法, 结合移动测试生成领域的实际任务场景, 充分利用大模型强化现有方法在文本输入和界面理解方面的能力.

(2) 挖掘更有效的测试预言

获得有效的测试预言是移动测试生成的核心挑战之一. 当前, 大部分测试生成方法只支持使用程序崩溃等被动预言作为测试预言, 只有少量工作能够主动生成涉及功能正确性等其他约束的测试预言. 例如, Su 等人[43]基于蜕变测试的思想提取相似执行路径的功能预言.

在该领域的未来研究中, 针对测试预言可以分别从加强被动预言和主动预言两个方向发展. 对于被动预言, 除了当前主要使用的崩溃之外, 可以挖掘其他可以作为预言的信息. 例如可以通过检测系统内存、电量流失速度、系统发热情况等挖掘程序的性能相关缺陷, 也可以通过蓝牙、NFC 等新型硬件设备的系统级事件设计新的预言. 对于主动预言, 可以挖掘更多语义相关的正确性约束和程序执行时的蜕变关系作为主动预言. 例如, 基于软件不变量分析技术[144]等传统软件缺陷定位领域常用的方法构建测试预言. 同时在当前, 以 ChatGPT 为代表的预训练大语言模型展现出良好的应用前景, 通过设计提示词, 基于大语言模型挖掘更丰富的程序执行语义信息, 从而使相关方法具备主动生成更多样化的测试预言的能力将会是未来一段时间的研究方向.

(3) 构建数据集和工具集

在本文梳理现有工作的过程中, 我们发现当前该领域验证过程中选取的被测应用和目标缺陷存在多样化的特点, 即不存在一个较为统一公认的大型数据集. 实验验证环节对于评价方法的有效性至关重要, 而基准数据集能够直接影响实验结论. 因此, 构建规模足够, 应用多样, 缺陷种类丰富的数据集是重要的研究方向. 另一方面, 虽然研究者提出了各种方法, 在比较过程中往往采取独立复现的方式, 存在一定的合法性威胁. 构建融合多种主流生成工具的统一平台能够缓解这一问题.

(4) 结合特殊任务场景

随着新的用户需求不断涌现, 移动应用 GUI 测试领域也不断出现新的特殊任务场景. 例如, 近年来被提出的专门面向残障人士的应用的测试方法, 就受到该领域研究者的广泛关注[12,37,145]. 另外, 各类传感器向着小型化、轻量化的发展, 越来越多的新型电子器件融入移动平台. 例如, 近年来诸如指纹实别、血氧和血压监测等传感器在移动设备中流行起来. 新的硬件设备将会提供更多的系统级事件和动作, 进而导致测试环境模型越来越复杂. 因此,

发掘、分析、提炼新的任务场景, 进一步优化抽象测试环境模型, 并针对新的特殊场景设计专门的测试生成方法, 是一个较有前景的研究方向.

(5) 推动性能验证过程标准化

当前测试生成工具在性能验证阶段广泛使用的评测指标包括测试生成器发现的缺陷数及对被测应用屏幕状态的覆盖情况. 虽然上述指标能够较为直观地反映测试生成器在特定被测应用集合上的表现, 但是仅使用这两种评测指标不足以全面, 准确地反映测试生成工具的实际性能. 这一局限性尤其表现在面向特殊任务场景的测试方法中. 另外, 当前不同研究者针对这两种指标的实际量化评估方法也不统一, 导致分析对比同类型测试工具性能较为困难. 值得注意的是, 已有若干研究者陆续关注了这一问题, 并提出了针对性能验证指标的优化方法. 如 Zhao 等人[131]提出的面向测试迁移工具的自动化性能评判工具 FrUITer, 以及 Lan 等人[136]针对覆盖率和检测缺陷数等常用评测指标展开的实证研究. 未来在性能评估阶段, 应针对不同测试任务分别定义统一的量化评估指标, 从而更精确地量化测试方法的实际性能, 同时方便后续研究者对已有方法进行性能对比分析等研究工作.

6 总 结

本文对移动应用用户界面测试自动生成技术进行了系统的分析, 提出了“测试生成器-测试环境”的研究框架, 并在该研究框架内从测试动作生成策略的角度将现有测试技术分为基于随机、基于启发式搜索、基于模型、基于机器学习、基于测试迁移这 5 类方法. 随后, 本文结合其他相关研究, 提出从缺陷类别、测试动作类别、测试动作序列生成策略类别这 3 个角度对相关工作进行分类整理. 本文还收集了重要的数据集和开源工具. 最后, 本文对该领域面临的挑战和未来研究方向进行了分析和总结.

References:

[1] Wang J, Jiang YY, Xu C, Ma XX, Lü J. Automatic test-input generation for Android applications. SCIENTIA SINICA Informationis, 2019, 49(10): 1234–1266 (in Chinese with English abstract). [doi: 10.1360/N112019-00003]

[2] Kong PF, Li L, Gao J, Liu K, Bissyande TF, Klein J. Automated testing of Android aps: A systematic literature review. IEEE Trans. on Reliability, 2019, 68(1): 45–66. [doi: 10.1109/TR.2018.2865733]

[3] Coppola R, Alégroth E. A taxonomy of metrics for GUI-based testing research: A systematic literature review. Information and Software Technology, 2022, 152: 107062. [doi: 10.1016/j.infsof.2022.107062]

[4] Tramontana P, Amalfitano D, Amatucci N, Fasolino AR. Automated functional testing of mobile applications: A systematic mapping study. Software Quality Journal, 2019, 27(1): 149–201. [doi: 10.1007/s11219-018-9418-6]

[5] Banerjee I, Nguyen B, Garousi V, Memon A. Graphical user interface (GUI) testing: Systematic mapping and repository. Information and Software Technology, 2013, 55(10): 1679–1694. [doi: 10.1016/j.infsof.2013.03.004]

[6] Zein S, Salleh N, Grundy J. A systematic mapping study of mobile application testing techniques. Journal of Systems and Software, 2016, 117: 334–356. [doi: 10.1016/j.jss.2016.03.065]

[7] Said KS, Nie LM, Ajibode AA, Zhou XY. GUI testing for mobile applications: Objectives, approaches and challenges. In: Proc. of the 12th Asia-Pacific Symp. on Internetware. Singapore: ACM, 2020. 51–60. [doi: 10.1145/3457913.3457931]

[8] Li C, Jiang YY, Xu C. GUI event-based record and replay technologies for Android Apps: A survey. Ruan Jian Xue Bao/Journal of Software, 2022, 33(5): 1612–1634 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6551.htm [doi: 10.13328/j.cnki.jos.006551]

[9] Zheng W, Tang H, Chen X, Zhang MQ, Xia X. State-of-the-art survey of compatibility test for Android mobile application. Journal of Computer Research and Development, 2022, 59(6): 1370–1387 (in Chinese with English abstract). [doi: 10.7544/issn1000-1239.20210105]

[10] Luo C, Goncalves J, Velloso E, Kostakos V. A survey of context simulation for testing mobile context-aware applications. ACM Computing Surveys, 2020, 53(1): 21. [doi: 10.1145/3372788]

[11] Chen K, Li YF, Chen YF, Fan CJ, Hu ZP, Yang W. GLIB: Towards automated test oracle for graphically-rich applications. In: Proc. of the 29th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Athens: ACM, 2021. 1093–1104. [doi: 10.1145/3468264.3468586]

[12] Salehnamadi N, Mehralian F, Malek S. Groundhog: An automated accessibility crawler for mobile Apps. In: Proc. of the 37th IEEE/ACM Int’l Conf. on Automated Software Engineering. Rochester: ACM, 2022. 50. [doi: 10.1145/3551349.3556905]

[13] Wang J, Jiang YY, Su T, Li SH, Xu C, Lu J, Su ZD. Detecting non-crashing functional bugs in Android Apps via deep-state differential

analysis. In: Proc. of the 30th ACM Joint European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Singapore: ACM, 2022. 434–446. [doi: 10.1145/3540250.3549170]

[14] Lin JW, Salehnamadi N, Malek S. Route: Roads not taken in UI testing. ACM Trans. on Software Engineering and Methodology, 2023, 32(3): 71. [doi: 10.1145/3571851]

[15] Coppola R, Ardito L, Torchiano M, Alégroth E. Translation from layout-based to visual Android test scripts: An empirical evaluation. Journal of Systems and Software, 2021, 171: 110845. [doi: 10.1016/j.jss.2020.110845]

[16] Machiry A, Tahiliani R, Naik M. Dynodroid: An input generation system for Android Apps. In: Proc. of the 9th Joint Meeting on Foundations of Software Engineering. Saint Petersburg: ACM, 2013. 224–234. [doi: 10.1145/2491411.2491450]

[17] Paydar S. Making Android Apps monkey-friendly. In: Proc. of the 7th IEEE/ACM Int'l Conf. on Mobile Software Engineering and Systems. Seoul: ACM, 2020. 16–20. [doi: 10.1145/3387905.3388609]

[18] Mao K, Harman M, Jia Y. Sapienz: Multi-objective automated testing for Android applications. In: Proc. of the 25th Int'l Symp. on Software Testing and Analysis. Saarbrücken: ACM, 2016. 94–105. [doi: 10.1145/2931037.2931054]

[19] Bose P, Das D, Vasan S, Mariani S, Grishchenko I, Continella A, Bianchi A, Kruegel C, Vigna G. Columbus: Android App testing through systematic callback exploration. In: Proc. of the 45th IEEE/ACM Int'l Conf. on Software Engineering. Melbourne: IEEE, 2023. 1381–1392. [doi: 10.1109/ICSE48619.2023.00121]

[20] Azim T, Neamtiu I. Targeted and depth-first exploration for systematic testing of Android Apps. In: Proc. of the 2013 ACM SIGPLAN Int'l Conf. on Object Oriented Programming Systems Languages & Applications. Indianapolis: ACM, 2013. 641–660. [doi: 10.1145/2509136.2509549]

[21] Mirzaei N, Bagheri H, Mahmood R, Malek S. SIG-Droid: Automated system input generation for Android applications. In: Proc. of the 26th IEEE Int'l Symp. on Software Reliability Engineering. Gaithersbury: IEEE, 2015. 461–471. [doi: 10.1109/ISSRE.2015.7381839]

[22] Gu TX, Sun CN, Ma XX, *et al*. Practical GUI testing of Android applications via model abstraction and refinement. In: Proc. of the 41st IEEE/ACM Int'l Conf. on Software Engineering. Montreal: IEEE, 2019. 269–280. [doi: 10.1109/ICSE.2019.00042]

[23] Behrang F, Orso A. Test migration between mobile Apps with similar functionality. In: Proc. of the 34th IEEE/ACM Int'l Conf. on Automated Software Engineering. San Diego: IEEE, 2019. 54–65. [doi: 10.1109/ASE.2019.00016]

[24] Qin X, Zhong H, Wang XY. TestMig: Migrating GUI test cases from iOS to Android. In: Proc. of the 28th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. Beijing: ACM, 2019. 284–295. [doi: 10.1145/3293882.3330575]

[25] Mariani L, Mohebbi A, Pezzè M, Terragni V. Semantic matching of GUI events for test reuse: Are we there yet? In: Proc. of the 30th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. Virtual: ACM, 2021. 177–190. [doi: 10.1145/3460319.3464827]

[26] Pan MX, Huang A, Wang GX, Zhang T, Li XD. Reinforcement learning based curiosity-driven testing of Android applications. In: Proc. of the 29th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. Virtual Event: ACM, 2020. 153–164. [doi: 10.1145/3395363.3397354]

[27] Hu G, Zhu LJ, Yang JF. AppFlow: Using machine learning to synthesize robust, reusable UI tests. In: Proc. of the 26th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Lake Buena Vista: ACM, 2018. 269–282. [doi: 10.1145/3236024.3236055]

[28] Zhao YX, Talebipour S, Baral K, Park H, Yee L, Khan SA, Brun Y, Medvidović N, Moran K. Avgust: Automating usage-based test generation from videos of App executions. In: Proc. of the 30th ACM Joint European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Singapore: ACM, 2022. 421–433. [doi: 10.1145/3540250.3549134]

[29] Liu Z, Chen CY, Wang JJ, Chen MZ, Wu BY, Che X, Wang DD, Wang Q. Make LLM a testing expert: Bringing human-like interaction to mobile GUI testing via functionality-aware decisions. In: Proc. of the 46th IEEE/ACM Int'l Conf. on Software Engineering. Lisbon: ACM, 2024. 100. [doi: 10.1145/3597503.3639180]

[30] Liu Z, Chen CY, Wang JJ, Che X, Huang YK, Hu J, Wang Q. Fill in the blank: Context-aware automated text input generation for mobile GUI testing. In: Proc. of the 45th IEEE/ACM Int'l Conf. on Software Engineering. Melbourne: IEEE, 2023. 1355–1367. [doi: 10.1109/ICSE48619.2023.00119]

[31] Choi W, Sen K, Necul G, Wang WY. DetReduce: Minimizing Android GUI test suites for regression testing. In: Proc. of the 40th IEEE/ACM Int'l Conf. on Software Engineering. Gothenburg: IEEE, 2018. 445–455. [doi: 10.1145/3180155.3180173]

[32] Wetzlmaier T, Ramler R. Hybrid monkey testing: Enhancing automated GUI tests with random test generation. In: Proc. of the 8th ACM SIGSOFT Int'l Workshop on Automated Software Testing. Paderborn: ACM, 2017. 5–10. [doi: 10.1145/3121245.3121247]

[33] Lelli V, Blouin A, Baudry B. Classifying and qualifying GUI defects. In: Proc. of the 8th IEEE Int'l Conf. on Software Testing, Verification and Validation. Graz: IEEE, 2015. 1–10. [doi: 10.1109/ICST.2015.7102582]

[34] Xiong YH, Xu MQ, Su T, Sun JL, Wang J, Wen H, Pu GG, He JF, Su ZD. An empirical study of functional bugs in Android Apps. In: Proc. of the 32nd ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. Seattle: ACM, 2023. 1319–1331. [doi: 10.1145/3597926.3598138]

[35] Liu Z, Wang JJ, Chen CY, Che X, Su YH, Wang Q. Empirical study on UI display issue detection in mobile applications. Ruan Jian Xue Bao/Journal of Software, 2024, 35(11): 5040–5064 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/7043.htm [doi: 10.13328/j.cnki.jos.007043]

[36] Li WJ, Jiang YY, Xu C, Liu YP, Ma XX, Lü J. Characterizing and detecting inefficient image displaying issues in Android Apps. In: Proc. of the 26th IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering. Hangzhou: IEEE, 2019. 355–365. [doi: 10.1109/SANER.2019.8668030]

[37] Eler MM, Rojas JM, Ge Y, Fraser G. Automated accessibility testing of mobile Apps. In: Proc. of the 11th IEEE Int'l Conf. on Software Testing, Verification and Validation. Västerås: IEEE, 2018. 116–126. [doi: 10.1109/ICST.2018.00021]

[38] Salehnamadi N, He ZY, Malek S. Assistive-technology aided manual accessibility testing in mobile Apps, powered by record-and-replay. In: Proc. of the 2023 CHI Conf. on Human Factors in Computing Systems. Hamburg: ACM, 2023. 73. [doi: 10.1145/3544548.3580679]

[39] Alshayban A, Malek S. AccessiText: Automated detection of text accessibility issues in Android Apps. In: Proc. of the 30th ACM Joint European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Singapore: ACM, 2022. 984–995. [doi: 10.1145/3540250.3549118]

[40] Zaeem RN, Prasad MR, Khurshid S. Automated generation of oracles for testing user-interaction features of mobile Apps. In: Proc. of the 7th IEEE Int'l Conf. on Software Testing, Verification and Validation. Cleveland: IEEE, 2014. 183–192. [doi: 10.1109/ICST.2014.31]

[41] Mariani L, Pezzè M, Zuddas D. Augusto: Exploiting popular functionalities for the generation of semantic GUI tests with oracles. In: Proc. of the 40th IEEE/ACM Int'l Conf. on Software Engineering. Gothenburg: IEEE, 2018. 280–290. [doi: 10.1145/3180155.3180162]

[42] Baral K, Johnson J, Mahmud J, Salma S, Fazzini M, Rubin J, Offutt J, Moran K. Automating GUI-based test oracles for mobile Apps. In: Proc. of the 21st Int'l Conf. on Mining Software Repositories. Lisbon: ACM, 2024. 309–321. [doi: 10.1145/3643991.3644930]

[43] Su T, Yan YC, Wang J, Sun JL, Xiong YH, Pu GG, Wang K, Su ZD. Fully automated functional fuzzing of Android Apps for detecting non-crashing logic bugs. Proc. of the ACM on Programming Languages, 2021, 5: 156. [doi: 10.1145/3485533]

[44] Xiong YH, Su T, Wang J, Sun JL, Pu GG, Su ZD. General and practical property-based testing for Android Apps. In: Proc. of the 39th ACM/IEEE Int'l Conf. on Automated Software Engineering. Sacramento: ACM, 2024. 53–64. [doi: 10.1145/3691620.3694986]

[45] Ardito L, Coppola R, Morisio M, Torchiano M. Espresso vs. EyeAutomate: An experiment for the comparison of two generations of Android GUI testing. In: Proc. of the 23rd Int'l Conf. on Evaluation and Assessment in Software Engineering. Copenhagen: ACM, 2019. 13–22. [doi: 10.1145/3319008.3319022]

[46] Coppola R, Ardito L, Torchiano M. Automated translation of Android context-dependent gestures to visual GUI test instructions. In: Proc. of the 12th Int'l Workshop on Automating TEST Case Design, Selection, and Evaluation. Athens: ACM, 2021. 17–24. [doi: 10.1145/3472672.3473954]

[47] Coppola R, Ardito L, Torchiano M, Alègroth E. Translation from visual to layout-based Android test cases: A proof of concept. In: Proc. of the 2020 IEEE Int'l Conf. on Software Testing, Verification and Validation Workshops. Porto: IEEE, 2020. 74–83. [doi: 10.1109/ICSTW50294.2020.00027]

[48] Yang S, Chen S, Fan LL, Xu SH, Hui ZW, Huang S. Compatibility issue detection for Android Apps based on path-sensitive semantic analysis. In: Proc. of the 45th IEEE/ACM Int'l Conf. on Software Engineering. Melbourne: IEEE, 2023. 257–269. [doi: 10.1109/ICSE48619.2023.00033]

[49] Auer M, Stahlbauer A, Fraser G. Android fuzzing: Balancing user-inputs and intents. In: Proc. of the 2023 IEEE Conf. on Software Testing, Verification and Validation. Dublin: IEEE, 2023. 37–48. [doi: 10.1109/ICST57152.2023.00013]

[50] Guo WN, Dong Z, Shen LW, Tian W, Su T, Peng X. Detecting and fixing data loss issues in Android Apps. In: Proc. of the 31st ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. Virtual: ACM, 2022. 605–616. [doi: 10.1145/3533767.3534402]

[51] Rahaman S, Farooq U, Neamtiu I, Zhao ZJ. Detecting potential user-data save & export losses due to Android App termination. In: Proc. of the 2023 IEEE/ACM Int'l Conf. on Automation of Software Test. Melbourne: IEEE, 2023. 152–162. [doi: 10.1109/AST58925.2023.00019]

[52] Fan LL, Su T, Chen S, Meng GZ, Liu Y, Xu LH, Pu GG, Su ZD. Large-scale analysis of framework-specific exceptions in Android Apps. In: Proc. of the 40th IEEE/ACM Int'l Conf. on Software Engineering. Gothenburg: IEEE, 2018. 408–419. [doi: 10.1145/3180155.3180222]

[53] Ravelo-Méndez W, Escobar-Velásquez C, Linares-Vásquez M. Kraken-mobile: Cross-device interaction-based testing of Android Apps. In: Proc. of the 2019 IEEE Int'l Conf. on Software Maintenance and Evolution. Cleveland: IEEE, 2019. 410–413. [doi: 10.1109/ICSME.2019.00071]

[54] Su T, Fan LL, Chen S, Liu Y, Xu LH, Pu GG, Su ZD. Why my App crashes? Understanding and benchmarking framework-specific exceptions of Android Apps. IEEE Trans. on Software Engineering, 2022, 48(4): 1115–1137. [doi: 10.1109/tse.2020.3013438]

[55] Wang WY, Li DF, Yang W, Cao YR, Zhang ZW, Deng YT, Xie T. An empirical study of Android test generation tools in industrial

cases. In: Proc. of the 33rd ACM/IEEE Int'l Conf. on Automated Software Engineering. Montpellier: IEEE, 2018. 738–748. [doi: 10.1145/3238147.3240465]

[56] Zheng HB, Li DF, Liang BH, Zeng X, Zheng WJ, Deng YT, Lam W, Yang W, Xie T. Automated test input generation for Android: Towards getting there in an industrial case. In: Proc. of the 39th IEEE/ACM Int'l Conf. on Software Engineering: Software Engineering in Practice Track. Buenos Aires: IEEE, 2017. 253–262. [doi: 10.1109/ICSE-SEIP.2017.32]

[57] Jr Borges NP, Gómez M, Zeller A. Guiding App testing with mined interaction models. In: Proc. of the 5th Int'l Conf. on Mobile Software Engineering and Systems. Gothenburg: ACM, 2018. 133–143. [doi: 10.1145/3197231.3197243]

[58] Yan JW, Zhou H, Deng X, Wang P, Yan RJ, Yan J, Zhang J. Efficient testing of GUI applications by event sequence reduction. Science of Computer Programming, 2021, 201: 102522. [doi: 10.1016/j.scico.2020.102522]

[59] Behrang F, Orso A. Seven reasons why: An in-depth study of the limitations of random test input generation for Android. In: Proc. of the 35th IEEE/ACM Int'l Conf. on Automated Software Engineering. Melbourne: ACM, 2020. 1066–1077. [doi: 10.1145/3324884.3416567]

[60] Hu H, Wang H, Dong RQ, Chen X, Chen CY. Enhancing GUI exploration coverage of Android Apps with deep link-integrated Monkey. ACM Trans. on Software Engineering and Methodology, 2024, 33(6): 163. [doi: 10.1145/3664810]

[61] Godboley S, Dalei D, Sadam R, Mohapatra DP. Agile GUI testing by computing novel mobile App coverage using Appium tool. In: Proc. of the 38th ACM/SIGAPP Symp. on Applied Computing. Tallinn: ACM, 2023. 1026–1029. [doi: 10.1145/3555776.3577806]

[62] Gross F, Fraser G, Zeller A. EXSYST: Search-based GUI testing. In: Proc. of the 34th Int'l Conf. on Software Engineering. Zurich: IEEE, 2012. 1423–1426. [doi: 10.1109/ICSE.2012.6227232]

[63] Vogel T, Tran C, Grunske L. A comprehensive empirical evaluation of generating test suites for mobile applications with diversity. Information and Software Technology, 2021, 130: 106436. [doi: 10.1016/j.infsof.2020.106436]

[64] Mao K, Harman M, Jia Y. Crowd intelligence enhances automated mobile testing. In: Proc. of the 32nd IEEE/ACM Int'l Conf. on Automated Software Engineering. Urbana: IEEE, 2017. 16–26. [doi: 10.1109/ASE.2017.8115614]

[65] Dong Z, Böhme M, Cojocaru L, Roychoudhury A. Time-travel testing of Android Apps. In: Proc. of the 42nd ACM/IEEE Int'l Conf. on Software Engineering. Seoul South: ACM, 2020. 481–492. [doi: 10.1145/3377811.3380402]

[66] Mahmood R, Mirzaei N, Malek S. Evodroid: Segmented evolutionary testing of Android Apps. In: Proc. of the 22nd ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. Hong Kong: IEEE, 2014. 599–609. [doi: 10.1145/2635868.2635896]

[67] Cheng L, Chang JL, Yang ZJ, Wang C. GUICat: GUI testing as a service. In: Proc. of the 31st IEEE/ACM Int'l Conf. on Automated Software Engineering. Singapore: ACM, 2016. 858–863. [doi: 10.1145/2970276.2970294]

[68] Salvesen K, Galeotti JP, Gross F, Fraser G, Zeller A. Using dynamic symbolic execution to generate inputs in search-based GUI testing. In: Proc. of the 8th IEEE/ACM Int'l Workshop on Search-based Software Testing. Florence: IEEE, 2015. 32–35. [doi: 10.1109/SBST.2015.15]

[69] Karlsson S, Čaušević A, Sundmark D, Larsson M. Model-based automated testing of mobile applications: An industrial case study. In: Proc. of the 2021 IEEE Int'l Conf. on Software Testing, Verification and Validation Workshops. Porto de Galinhas: IEEE, 2021. 130–137. [doi: 10.1109/ICSTW52544.2021.00033]

[70] Baek YM, Bae DH. Automated model-based Android GUI testing using multi-level GUI comparison criteria. In: Proc. of the 31st IEEE/ACM Int'l Conf. on Automated Software Engineering. Singapore: ACM, 2016. 238–249. [doi: 10.1145/2970276.2970313]

[71] Lai DL, Rubin J. Goal-driven exploration for Android applications. In: Proc. of the 34th IEEE/ACM Int'l Conf. on Automated Software Engineering. San Diego: IEEE, 2019. 115–127. [doi: 10.1109/ASE.2019.00021]

[72] Guo WN, Shen LW, Su T, Peng X, Xie WY. Improving automated GUI exploration of Android Apps via static dependency analysis. In: Proc. of the 2020 IEEE Int'l Conf. on Software Maintenance and Evolution. Adelaide: IEEE, 2020. 557–568. [doi: 10.1109/ICSME46990.2020.00059]

[73] Nguyen BN, Robbins B, Banerjee I, Memon A. GUITAR: An innovative tool for automated testing of GUI-driven software. Automated Software Engineering, 2014, 21(1): 65–105. [doi: 10.1007/s10515-013-0128-9]

[74] Amalfitano D, Fasolino AR, Tramontana P, Ta BD, Memon AM. MobiGUITAR: Automated model-based testing of mobile Apps. IEEE Software, 2015, 32(5): 53–59. [doi: 10.1109/MS.2014.55]

[75] Su T, Meng GZ, Chen YT, Wu K, Yang WM, Yao Y, Pu GG, Liu Y, Su ZD. Guided, stochastic model-based GUI testing of Android Apps. In: Proc. of the 11th Joint Meeting on Foundations of Software Engineering. Paderborn: ACM, 2017. 245–256. [doi: 10.1145/3106237.3106298]

[76] Wu SH, Li JF, Zhou H, Fang YS, Zhao KF, Wang HY, Qian CX, Luo XP. CydiOS: A model-based testing framework for iOS Apps. In: Proc. of the 32nd ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. Seattle: ACM, 2023. 1–13. [doi: 10.1145/3597926.3598033]

[77] Cao YZ, Wu GQ, Chen W, Wei J. CrawlDroid: Effective model-based GUI testing of Android Apps. In: Proc. of the 10th Asia-Pacific

Symp. on Internetware. Beijing: ACM, 2018. 19. [doi: 10.1145/3275219.3275238]

[78] Zhang XY, Fan LL, Chen S, Su YC, Li BY. Scene-driven exploration and GUI modeling for Android Apps. In: Proc. of the 38th IEEE/ACM Int'l Conf. on Automated Software Engineering. Luxembourg: IEEE, 2023. 1251–1262. [doi: 10.1109/ASE56229.2023.00179]

[79] Yu SC, Fang CR, Du MZ, Ding ZM, Chen ZY, Su ZD. Practical, automated scenario-based mobile App testing. IEEE Trans. on Software Engineering, 2024, 50(7): 1949–1966. [doi: 10.1109/TSE.2024.3414672]

[80] Zhang Z, Fazle M, Komei R, *et al*. Mobile bug report reproduction via global search on the App UI model. In: Proc. of the 32nd ACM Joint European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. 2024. [doi: 10.1145/3660824]

[81] Cheng HL, Tang EY, Yu CZ, Zhang CC, Chen X, Wang LZ, Bu L, Li XD. Approach of sketch-guided GUI testing for mobile App. Ruan Jian Xue Bao/Journal of Software, 2020, 31(12): 3671–3684 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5873.htm [doi: 10.13328/j.cnki.jos.005873]

[82] Sun JL, Su T, Sun J, Li JW, Wang MF, Pu GG. Property-based testing for validating user privacy-related functionalities in social media Apps. In: Companion Proc. of the 32nd ACM Int'l Conf. on the Foundations of Software Engineering. Porto de Galinhas: ACM, 2024. 440–451. [doi: 10.1145/3663529.3663863]

[83] Peng C, Rajan A, Cai TQ. CAT: Change-focused Android GUI testing. In: Proc. of the 2021 IEEE Int'l Conf. on Software Maintenance and Evolution. Luxembourg: IEEE, 2021. 460–470. [doi: 10.1109/ICSME52107.2021.00047]

[84] Ngo CD, Pastore F, Briand L. Automated, cost-effective, and update-driven App testing. ACM Trans. on Software Engineering and Methodology, 2022, 31(4): 61. [doi: 10.1145/3502297]

[85] Ghorbani N, Jabbarvand R, Salehnamadi N, Garcia J, Malek S. DeltaDroid: Dynamic delivery testing in Android. ACM Trans. on Software Engineering and Methodology, 2023, 32(4): 84. [doi: 10.1145/3563213]

[86] Fazzini M, Orso A. Automated cross-platform inconsistency detection for mobile Apps. In: Proc. of the 32nd IEEE/ACM Int'l Conf. on Automated Software Engineering. Urbana: IEEE, 2017. 308–318. [doi: 10.1109/ASE.2017.8115644]

[87] Ge XT, Yu SC, Fang CR, Zhu Q, Zhao ZH. Leveraging Android automated testing to assist crowdsourced testing. IEEE Trans. on Software Engineering, 2023, 49(4): 2318–2336. [doi: 10.1109/TSE.2022.3216879]

[88] Li YY, Feng Y, Hao R, Chen ZY. Human-machine collaborative testing for Android applications. In: Proc. of the 23rd IEEE Int'l Conf. on Software Quality, Reliability, and Security. Chiang Mai: IEEE, 2023. 440–451. [doi: 10.1109/QRS60937.2023.00050]

[89] Li YY, Feng Y, Guo C, Chen ZY, Xu BW. Crowdsourced test case generation for Android applications via static program analysis. Automated Software Engineering, 2023, 30(2): 26. [doi: 10.1007/s10515-023-00394-w]

[90] Fang CR, Yu SC, Su T, Zhang J, Tian YH, Liu Y. Test report generation for Android App testing via heterogeneous data analysis. IEEE Trans. on Software Engineering, 2023, 49(5): 3032–3051. [doi: 10.1109/TSE.2023.3237247]

[91] Zhang X, Feng C, Lei J, Tang CJ. Real time idle state detection method in fuzzing test in GUI program. Ruan Jian Xue Bao/Journal of Software, 2018, 29(5): 1288–1302 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5493.htm [doi: 10.13328/j.cnki.jos.005493]

[92] Zhao Y, Tang EY, Cai HP, Guo X, Wang XY, Meng N. A lightweight approach of human-like playtest for Android Apps. In: Proc. of the 2022 IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering. Honolulu: IEEE, 2022. 309–320. [doi: 10.1109/SANER53432.2022.00047]

[93] Moreira RMLM, Paiva ACR. PBGT tool: An integrated modeling and testing environment for pattern-based GUI testing. In: Proc. of the 29th ACM/IEEE Int'l Conf. on Automated Software Engineering. Vasteras: ACM, 2014. 863–866. [doi: 10.1145/2642937.2648618]

[94] Ramler R, Klammer C, Wetzlmaier T. Lessons learned from making the transition to model-based GUI testing. In: Proc. of the 10th ACM SIGSOFT Int'l Workshop on Automating TEST Case Design, Selection, and Evaluation. Tallinn: ACM, 2019. 22–27. [doi: 10.1145/3340433.3342823]

[95] Wang WY, Yang W, Xu TY, Xie T. VET: Identifying and avoiding UI exploration tarpits. In: Proc. of the 29th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Athens: ACM, 2021. 83–94. [doi: 10.1145/3468264.3468554]

[96] Koroglu Y, Sen A, Muslu O, Mete Y, Ulker C, Tanriverdi T, Donmez Y. QBE: QLearning-based exploration of Android applications. In: Proc. of the 11th IEEE Int'l Conf. on Software Testing, Verification and Validation. Västerås: IEEE, 2018. 105–115. [doi: 10.1109/ICST.2018.00020]

[97] Adamo D, Khan MK, Koppula S, Bryce R. Reinforcement learning for Android GUI testing. In: Proc. of the 9th ACM SIGSOFT Int'l Workshop on Automating TEST Case Design, Selection, and Evaluation. Lake Buena Vista: ACM, 2018. 2–8. [doi: 10.1145/3278186.3278187]

[98] Degott C, Jr Borges NP, Zeller A. Learning user interface element interactions. In: Proc. of the 28th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. Beijing: ACM, 2019. 296–306. [doi: 10.1145/3293882.3330569]

[99] Lv ZW, Peng C, Zhang Z, Su T, Liu K, Yang P. Fastbot2: Reusable automated model-based GUI testing for Android enhanced by reinforcement learning. In: Proc. of the 37th IEEE/ACM Int'l Conf. on Automated Software Engineering. Rochester: ACM, 2022. 135. [doi: 10.1145/3551349.3559505]

[100] Koroglu Y, Sen A. Functional test generation from UI test scenarios using reinforcement learning for Android applications. Software Testing, Verification and Reliability, 2021, 31(3): e1752. [doi: 10.1002/stvr.1752]

[101] Yu SC, Fang CR, Li X, Ling YC, Chen ZY, Su ZD. Effective, platform-independent GUI testing via image embedding and reinforcement learning. ACM Trans. on Software Engineering and Methodology, 2024, 33(7): 175. [doi: 10.1145/3674728]

[102] YazdaniBanafsheDaragh F, Malek S. Deep GUI: Black-box GUI input generation with deep learning. In: Proc. of the 36th IEEE/ACM Int'l Conf. on Automated Software Engineering. Melbourne: IEEE, 2021. 905–916. [doi: 10.1109/ASE51524.2021.9678778]

[103] Lan YH, Lu YF, Li Z, Pan MX, Yang WH, Zhang T, Li XD. Deeply Reinforcing Android GUI testing with deep reinforcement learning. In: Proc. of the 46th IEEE/ACM Int'l Conf. on Software Engineering. Lisbon: ACM, 2024. 71. [doi: 10.1145/3597503.3623344]

[104] Zhao Y, Harrison B, Yu TT. DinoDroid: Testing Android Apps using deep Q-networks. ACM Trans. on Software Engineering and Methodology, 2024, 33(5): 122. [doi: 10.1145/3652150]

[105] Peng C, Lv ZW, Fu JR, Liang JY, Zhang Z, Rajan A, Yang P. Hawkeye: Change-targeted testing for Android Apps based on deep reinforcement learning. In: Proc. of the 46th Int'l Conf. on Software Engineering: Software Engineering in Practice. Lisbon: ACM, 2024. 298–308. [doi: 10.1145/3639477.3639749]

[106] Zhang SK, Li YC, Lei HW, Jiang P, Li D, Guo Y, Chen XQ. GUI fuzzing framework for mobile Apps based on multi-modal representation. Ruan Jian Xue Bao/Journal of Software, 2024, 35(7): 3162–3179 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/7106.htm [doi: 10.13328/j.cnki.jos.007106]

[107] Ran DZ, Wang H, Wang WY, Xie T. Badge: Prioritizing UI events with hierarchical multi-armed bandits for automated UI testing. In: Proc. of the 45th IEEE/ACM Int'l Conf. on Software Engineering. Melbourne: IEEE, 2023. 894–905. [doi: 10.1109/ICSE48619.2023.00083]

[108] Liu P, Zhang XY, Pistoia M, Zheng YH, Marques M, Zeng LF. Automatic text input generation for mobile testing. In: Proc. of the 39th IEEE/ACM Int'l Conf. on Software Engineering. Buenos Aires: IEEE, 2017. 643–653. [doi: 10.1109/ICSE.2017.65]

[109] White TD, Fraser G, Brown GJ. Improving random GUI testing with image-based widget detection. In: Proc. of the 28th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. Beijing: ACM, 2019. 307–317. [doi: 10.1145/3293882.3330551]

[110] Feng SD, Xie ML, Chen CY. Efficiency matters: Speeding up automated testing with GUI rendering inference. In: Proc. of the 45th IEEE/ACM Int'l Conf. on Software Engineering. Melbourne: IEEE, 2023. 906–918. [doi: 10.1109/ICSE48619.2023.00084]

[111] Li YC, Yang ZY, Guo Y, Chen XQ. Humanoid: A deep learning-based approach to automated black-box Android App testing. In: Proc. of the 34th IEEE/ACM Int'l Conf. on Automated Software Engineering. San Diego: IEEE, 2019. 1070–1073. [doi: 10.1109/ASE.2019.00104]

[112] Nayak V, Kraus D. Session-based recommender systems for action selection in GUI test generation. In: Proc. of the 2020 IEEE Int'l Conf. on Software Testing, Verification and Validation Workshops. Porto: IEEE, 2020. 372–375. [doi: 10.1109/ICSTW50294.2020.00066]

[113] Hu YX, Gu JZ, Hu SQ, Zhang Y, Tian WJ, Guo SY, Chen CY, Zhou YF. Appaction: Automatic GUI interaction for mobile Apps via holistic widget perception. In: Proc. of the 31st ACM Joint European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. San Francisco: ACM, 2023. 1786–1797. [doi: 10.1145/3611643.3613885]

[114] Saha A, Song Y, Mahmud J, Zhou Y, Moran K, Chaparro O. Toward the automated localization of buggy mobile App UIs from bug descriptions. arXiv:2408.04075, 2024.

[115] Liu J, He DJ, Wu DY, Xue JL. Correlating UI contexts with sensitive API calls: Dynamic semantic extraction and analysis. In: Proc. of the 31st IEEE Int'l Symp. on Software Reliability Engineering. Coimbra: IEEE, 2020. 241–252. [doi: 10.1109/ISSRE5003.2020.00031]

[116] Peng C, Zhang Z, Lv ZW, *et al*. MUBot: Learning to test large-scale commercial Android Apps like a human. In: Proc. of the 2022 IEEE Int'l Conf. on Software Maintenance and Evolution. IEEE, 2022. 543–552. [doi: 10.1109/ICSME55016.2022.00074]

[117] Zimmermann D, Koziolek A. Automating GUI-based software testing with GPT-3. In: Proc. of the 2023 IEEE Int'l Conf. on Software Testing, Verification and Validation Workshops. Dublin: IEEE, 2023. 62–65. [doi: 10.1109/ICSTW58534.2023.00022]

[118] Taeb M, Swearngin A, Schoop E, Cheng RJ, Jiang Y, Nichols J. AXNav: Replaying accessibility tests from natural language. In: Proc. of the 2024 CHI Conf. on Human Factors in Computing Systems. Honolulu: ACM, 2024. 962. [doi: 10.1145/3613904.3642777]

[119] Yoon J, Feldt R, Yoo S. Intent-driven mobile GUI testing with autonomous Large Language Model agents. In: Proc. of the 2024 IEEE Conf. on Software Testing, Verification and Validation. Toronto: IEEE, 2024. 129–139. [doi: 10.1109/ICST60714.2024.00020]

[120] Wang DB, Zhao Y, Feng SD, Zhang ZX, Halfond WGJ, Chen CY, Sun XX, Shi JF, Yu TT. Feedback-driven automated whole bug report reproduction for Android Apps. arXiv:2407.05165, 2024.

[121] Liu Z, Chen CY, Wang JJ, Chen MZ, Wu BY, Tian ZL, Huang YK, Hu J, Wang Q. Testing the limits: Unusual text inputs generation

for mobile App crash detection with large language model. In: Proc. of the 46th IEEE/ACM Int'l Conf. on Software Engineering. Lisbon: ACM, 2024. 137. [doi: 10.1145/3597503.3639118]

[122] Behrang F, Orso A. Test migration for efficient large-scale assessment of mobile App coding assignments. In: Proc. of the 27th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. Amsterdam: ACM, 2018. 164–175. [doi: 10.1145/3213846.3213854]

[123] Yu SC, Fang CR, Yun YX, Feng Y. Layout and image recognition driving cross-platform automated mobile testing. In: Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering. Madrid: IEEE, 2021. 1561–1571. [doi: 10.1109/ICSE43902.2021.00139]

[124] Zhang YK, Zhang WJ, Ran DZ, Zhu QH, Dou CF, Hao D, Xie T, Zhang L. Learning-based widget matching for migrating GUI test cases. In: Proc. of the 46th IEEE/ACM Int'l Conf. on Software Engineering. Lisbon: ACM, 2024. 69. [doi: 10.1145/3597503.3623322]

[125] Ji RH, Zhu TW, Zhu XQ, Chen CY, Pan MX, Zhang T. Vision-based widget mapping for test migration across mobile platforms: Are we there yet? In: Proc. of the 38th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Luxembourg: IEEE, 2023. 1416–1428. [doi: 10.1109/ASE56229.2023.00068]

[126] Lin JW, Malek S. GUI test transfer from Web to Android. In: Proc. of the 2022 IEEE Conf. on Software Testing, Verification and Validation. Valencia: IEEE, 2022. 1–11. [doi: 10.1109/ICST53961.2022.00011]

[127] Lin JW, Jabbarvand R, Malek S. Test transfer across mobile Apps through semantic mapping. In: Proc. of the 34th IEEE/ACM Int'l Conf. on Automated Software Engineering. San Diego: IEEE, 2019. 42–53. [doi: 10.1109/ASE.2019.00015]

[128] Mariani L, Pezzè M, Terragni V, Zuddas D. An evolutionary approach to adapt tests across mobile Apps. In: Proc. of the 2021 IEEE/ACM Int'l Conf. on Automation of Software Test. Madrid: IEEE, 2021. 70–79. [doi: 10.1109/AST52587.2021.00016]

[129] Zhang YK, Zhu QH, Yang JW, Liu C, Zhang WJ, Zhao YF, Hao D, Zhang L. Synthesis-based enhancement for GUI test case migration. In: Proc. of the 33rd ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. Vienna: ACM, 2024. 869–881. [doi: 10.1145/3650212.3680327]

[130] Jha AK, Kim DY, Lee WJ. A framework for testing Android Apps by reusing test cases. In: Proc. of the 6th IEEE/ACM Int'l Conf. on Mobile Software Engineering and Systems. Montreal: IEEE, 2019. 20–24. [doi: 10.1109/MOBILESoft.2019.00012]

[131] Zhao YX, Chen J, Sejfia A, Laser MS, Zhang J, Sarro F, Harman M, Medvidovic N. FrUITeR: A framework for evaluating UI test reuse. In: Proc. of the 28th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Virtual Event: ACM, 2020. 1190–1201. [doi: 10.1145/3368089.3409708]

[132] Fulcini T, Coppola R, Torchiano M, Ardito L. An analysis of widget layout attributes to support Android GUI-based testing. In: Proc. of the 2023 IEEE Int'l Conf. on Software Testing, Verification and Validation Workshops. Dublin: IEEE, 2023. 117–125. [doi: 10.1109/ICSTW58534.2023.00033]

[133] Thung F, Irsan IC, Liu JK, Lo D. Towards benchmarking the coverage of automated testing tools in Android against manual testing. In: Proc. of the 11th IEEE/ACM Int'l Conf. on Mobile Software Engineering and Systems. Lisbon: ACM, 2024. 74–77. [doi: 10.1145/3647632.3651394]

[134] Cui CH, Li T, Wang JJ, Chen CY, Towey D, Huang RB. Large Language Models for mobile GUI text input generation: An empirical study. arXiv:2404.08948, 2024.

[135] Zhong Y, Shi MY, Fang CR, Zhao ZH, Chen ZY. Towards comprehensive evaluation for Android automated testing tools. Ruan Jian Xue Bao/Journal of Software, 2023, 34(4): 1630–1649 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6701.htm [doi: 10.13328/j.cnki.jos.006701]

[136] Lan YH, Lu YF, Pan MX, Li XD. Navigating mobile testing evaluation: A comprehensive statistical analysis of Android GUI testing metrics. In: Proc. of the 39th ACM/IEEE Int'l Conf. on Automated Software Engineering. Sacramento: ACM, 2024. 944–956. [doi: 10.1145/3691620.3695476]

[137] Linares-Vásquez M, Moran K, Poshyvanyk D. Continuous, evolutionary and large-scale: A new perspective for automated mobile App testing. In: Proc. of the 2017 IEEE Int'l Conf. on Software Maintenance and Evolution. Shanghai: IEEE, 2017. 399–410. [doi: 10.1109/ICSME.2017.27]

[138] Choudhary SR, Gorla A, Orso A. Automated test input generation for Android: Are we there yet? In: Proc. of the 30th IEEE/ACM Int'l Conf. on Automated Software Engineering. Lincoln: IEEE, 2015. 429–440. [doi: 10.1109/ASE.2015.89]

[139] Su T, Wang J, Su ZD. Benchmarking automated GUI testing for Android against real-world bugs. In: Proc. of the 29th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Athens: ACM, 2021. 119–130. [doi: 10.1145/3468264.3468620]

[140] Linares-Vásquez M, Bernal-Cárdenas C, Moran K, Poshyvanyk D. How do developers test Android applications? In: Proc. of the 2017 IEEE Int'l Conf. on Software Maintenance and Evolution. Shanghai: IEEE, 2017. 613–622. [doi: 10.1109/ICSME.2017.47]

[141] Lin JW, Salehnamadi N, Malek S. Test automation in open-source Android Apps: A large-scale empirical study. In: Proc. of the 35th IEEE/ACM Int'l Conf. on Automated Software Engineering. Virtual Event: ACM, 2020. 1078–1089. [doi: 10.1145/3324884.3416623]

[142] Coppola R, Morisio M, Torchiano M. Scripted GUI testing of Android Apps: A study on diffusion, evolution and fragility. In: Proc. of

the 13th Int'l Conf. on Predictive Models and Data Analytics in Software Engineering. Toronto: ACM, 2017. 22–32. [doi: 10.1145/3127005.3127008]

[143] Sun JL, Su T, Li JX, Dong Z, Pu GG, Xie T, Su ZD. Understanding and finding system setting-related defects in Android Apps. In: Proc. of the 30th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. Virtual: ACM, 2021. 204–215. [doi: 10.1145/3460319.3464806]

[144] Wang B, Lu SR, Jiang JJ, Xiong YF. Survey of dynamic analysis based program invariant synthesis techniques. Ruan Jian Xue Bao/Journal of Software, 2020, 31(6): 1681–1702 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6014.htm [doi: 10.13328/j.cnki.jos.006014]

[145] Salehnamadi N, Alshayban A, Lin JW, Ahmed I, Branham S, Malek S. Latte: Use-case and assistive-service driven automated accessibility testing framework for Android. In: Proc. of the 2021 CHI Conf. on Human Factors in Computing Systems. Yokohama: ACM, 2021. 274. [doi: 10.1145/3411764.3445455]

附中文参考文献:

[1] 王珏, 蒋炎岩, 许畅, 马晓星, 吕建. Android 应用测试输入自动生成技术. 中国科学: 信息科学, 2019, 49(10): 1234–1266. [doi: 10.1360/N112019-00003]

[8] 李聪, 蒋炎岩, 许畅. 基于 GUI 事件的安卓应用录制重放关键技术综述. 软件学报, 2022, 33(5): 1612–1634. http://www.jos.org.cn/1000-9825/6551.htm [doi: 10.13328/j.cnki.jos.006551]

[9] 郑炜, 唐辉, 陈翔, 张满青, 夏鑫. 安卓移动应用兼容性测试综述. 计算机研究与发展, 2022, 59(6): 1370–1387. [doi: 10.7544/issn1000-1239.20210105]

[35] 刘哲, 王俊杰, 陈春阳, 车行, 苏宇辉, 王青. 移动应用程序中用户界面显示缺陷检测的经验研究. 软件学报, 2024, 35(11): 5040–5064. http://www.jos.org.cn/1000-9825/7043.htm [doi: 10.13328/j.cnki.jos.007043]

[81] 成浩亮, 汤恩义, 玉淳舟, 张初成, 陈鑫, 王林章, 卜磊, 李宣东. 一种手绘制导的移动应用界面测试方法. 软件学报, 2020, 31(12): 3671–3684. http://www.jos.org.cn/1000-9825/5873.htm [doi: 10.13328/j.cnki.jos.005873]

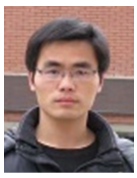
[91] 张兴, 冯超, 雷菁, 唐朝京. 一种面向模糊测试的 GUI 程序空转状态实时检测方法. 软件学报, 2018, 29(5): 1288–1302. http://www.jos.org.cn/1000-9825/5493.htm [doi: 10.13328/j.cnki.jos.005493]

[106] 张少坤, 李元春, 雷瀚文, 蒋鹏, 李锭, 郭耀, 陈向群. 基于多模态表征的移动应用 GUI 模糊测试框架. 软件学报, 2024, 35(7): 3162–3179. http://www.jos.org.cn/1000-9825/7106.htm [doi: 10.13328/j.cnki.jos.007106]

[135] 钟怡, 石孟雨, 房春荣, 赵志宏, 陈振宇. 面向安卓自动化测试工具综合评估. 软件学报, 2023, 34(4): 1630–1649. http://www.jos.org.cn/1000-9825/6701.htm [doi: 10.13328/j.cnki.jos.006701]

[144] 王博, 卢思睿, 姜佳君, 熊英飞. 基于动态分析的软件不变量综合技术. 软件学报, 2020, 31(6): 1681–1702. http://www.jos.org.cn/1000-9825/6014.htm [doi: 10.13328/j.cnki.jos.006014]

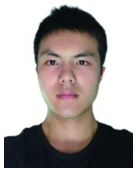
王博(1989－), 男, 博士, 讲师, CCF 专业会员, 主要研究领域为软件分析, 软件测试.



董震(1987－), 男, 博士, 副研究员, CCF 专业会员, 主要研究领域为程序分析, 软件测试, 软件工程.

陈冲(2000－), 男, 硕士生, CCF 学生会员, 主要研究领域为软件测试.

林友芳(1971－), 男, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为数据挖掘, 大数据工程与技术.



邓明(2002－), 男, 硕士生, CCF 学生会员, 主要研究领域为软件测试.

郝丹(1979－), 女, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为软件工程, 软件测试.