测试输入生成

陶泽华[1]

[1](南京大学 软件学院,江苏 南通 191250133)
通讯作者: 陶泽华, E-mail: 191250133@smail.nju.edu.cn

摘 要: 近年来，移动应用发展的速度激增，几乎每天都会有新的移动应用程序上架。但是要保证它们的质量是一项严峻的挑战，不充分的测试通常会导致应用程序出现 bug。手动化测试的覆盖率无法满足测试的要求，且需要大量的人力，因此自动化测试应运而生，且变得越来越流行。目前自动化测试有三大主流策略：（1）基于随机的自动化测试（2）基于模型的自动化测试（3）基于学习的自动化测试。但是这三大主流的自动化测试策略都面临着一个共同的挑战。目前市场上存在不同的操作系统、不同的 Android 版本、不同的应用版本和多种多样的应用程序，自动化测试工具很难适应每一种操作系统、Android 版本和应用，因此往往具有局限性。提高测试工具的兼容性和测试用例的可重用性就显得尤为重要。

关键词: 移动应用自动化测试、测试输入生成、随机测试、模型、强化学习、兼容性测试

近年来，移动应用变得无处不在，数量急剧增加。统计数据[1]显示，谷歌 Play 每月有超过 5 万款新的 Android 应用被提交。应用开发者们面对市场上激烈的竞争，他们通常面临着在最后期限完成发行应用的巨大压力。然而，要保证它们的质量是一项挑战。不充分的测试通常会导致应用程序出现错误。目前有两种主流的移动测试方法：手动测试和自动化测试。在手动测试中，测试人员通过人工分析应用 UI 界面，根据自己的判断生成测试用例并执行测试。然而这种方法需要大量的人力，因为测试人员需要在整个测试期间与应用程序密切互动。此外，人工生成测试用例进行测试时测试人员常常会错过可能触发异常的不常见用例，这就导致了不充分、不完全的移动应用测试。另外，由于用户界面(UI)作为用户交互的主要媒介，是一个很好的测试切入点。UI 通常通过统一的接口公开大量的功能，这使得它们非常适合自动化测试。虽然手动和脚本 UI 测试是一种常见的实践，但自动化 UI 测试正变得越来越流行。自动化测试可以使开发人员和测试人员很容易地在任何时候以及在许多设备上的多个应用程序上运行自动化的 UI 测试。测试输入的自动生成覆盖了更加完整的输入空间，在提高覆盖率、降低脚本复杂度的同时发现最多数量的错误。

目前的自动化测试有三大主流策略，分别是基于随机的自动化测试、基于模型的自动化测试和基于学习的自动化测试。基于随机的自动化测试采用随机策略为 Android 应用生成输入。Monkey 是由 Google 提出的最常用的 Android 测试工具，是这种策略的一个典型例子。它通过与屏幕坐标的随机交互，生成用户事件的伪随机流。这个基本的随机策略在一些基准应用上执行得相当好。然而，生成的测试用例包含大量无效或冗余的事件，这将威胁到测试的有效性。此外，测试是不平衡的，一些难以达到的功能可能永远不会被探索。基于模型的策略根据静态或动态构建的应用程序模型来描述应用程序的行为，然后从模型中派生测试用例来发现 bug[2]。由于测试用例是在构建模型的基础上生成的，它的准确性和完整性将是非常重要的。基于强化学习的测试通过采用激励机制。若在测试中某个操作探索到了未探索的空间，激励机制会给这个操作更高的激励值。测试会根据激励值的变化来不断的调整它的探索策略，这样测试就能涵盖到应用程序的更多功能，使得测试更加充分。

目前市场上存在不同的操作系统、不同的 Android 版本、不同的应用版本和多种多样的应用程序，而目前

的大部分自动化测试工具很难适用于不同的操作系统、Android 版本和应用程序。因此迫切需要加强的就是提高测试工具的兼容性和测试用例的可重用性，这也是所有的自动化测试工具需要解决的一个共同的问题。

本文第一、二、三节分别介绍了自动化生成测试输入的三大主流策略，并且给出了该种策略下具有代表性的工具的介绍。第四节就所有自动化生成测试输入工具面对的共同问题进行探讨。第五节对全文进行总结分析。

1 基于随机的自动化测试

基于随机的自动化测试顾名思义就是在测试时随机的动作序列，具体的来讲就是在移动应用的 UI 界面上随机点击按钮，生成测试输入以进行测试。最具有代表性的随机自动化测试工具就是 Google 公司开发的 Monkey 工具，它被提出以减少人工工作和最大化用例覆盖率。

1.1 Monkey

Monkey 是一个纯粹的随机测试工具[3]。“Monkey”是一个比喻，用来描述这种类型的测试是如何工作的，简单来说就像猴子一样，该工具执行随机的动作序列，包括点击 UI 视图(或 UI 屏幕)上的按钮，并执行随机击键。原则上，Monkey 会发出伪随机的 UI 事件流和一些系统事件，它可以在没有任何指导的情况下生成纯随机事件作为测试输入，它可以生成手动测试时忽视掉的用例。但是 Monkey 不能保证引导测试统一地遍历 GUI，这通常会导致低活动覆盖率。另外 Monkey 随机生成的事件是低级的，并且通常是一个很长的序列，这使得后续的调试变得非常复杂[4]。Monkey 在业界被广泛应用于压力测试，因为它易于使用，并与任何 Android 版本兼容，因此许多新技术都会与 Monkey 进行比较，像 Stoat[5]、APE[4]、Sapienz[6]。然而这些先进的自动化测试工具在给定的适当的运行时间下的测试效果是比不上 Monkey 的。

为了提高 Monkey 测试的覆盖率，研究人员提出了多种新的搜索策略进行改进。尽管有了这些改进，但在 Monkey 测试期间提供适当的文本输入仍然是一个突出的障碍，这阻碍了 Monkey 测试方法的大规模采用[7]。在许多用例中，大多数现有技术几乎无法提供有意义的文本输入。例如，对于一个地图应用来说，Monkey 测试很难提供“南京”那样有意义的地名的输入。相反，它将产生无关的输入。因此，搜索将无法找到任何结果，从而使 Monkey 无法进入显示适当结果的屏幕，测试就无法继续下去。手工规范输入可以缓解这个问题，但是需要大量的人力。刘鹏[7]等人提出了一个解决方案，让 Monkey 自动生成相关的文本输入。有了这些输入，Monkey 就可以通过较长的动作序列深入到新的 UI 屏幕，而不是在一开始就卡住无法继续测试下去。他们在实现中使用 RNN 模型来预测给定上下文中的文本输入值。此外，由于预测输入的应用程序与训练模型中的应用程序的不同，研究者们使用 Word2Vec 模型改进了 RNN 模型，该模型有助于识别语义相似的上下文，尽管它们的句法形式不同。换句话说，Word2Vec 模型可以使输入生成独立于应用程序。如果在测试时遇到了一个在训练阶段没有出现过的单词，利用 Word2Vec 模型就可以将它与训练阶段出现过的类似单词联系起来，此时再使用 RNN 模型就可以精准的预测测试所需要的文本输入。

2 基于模型的自动化测试

基于模型的测试(MBT)是自动化测试的另一种流行方法，它通过静态或动态分析应用程序行为抽象构建出模型，然后从模型中派生出测试用例来验证应用程序。因为测试用例完全是在模型的基础上生成的，所以构建的模型的准确性和完整性非常重要。

2.1 Stoat

苏婷[3]等人提出了一种新的基于随机模型的测试化方法 Stoat（stochastic model App Tester）。Stoat 的执行分为模型构建阶段、模型变异、测试生成和执行阶段。

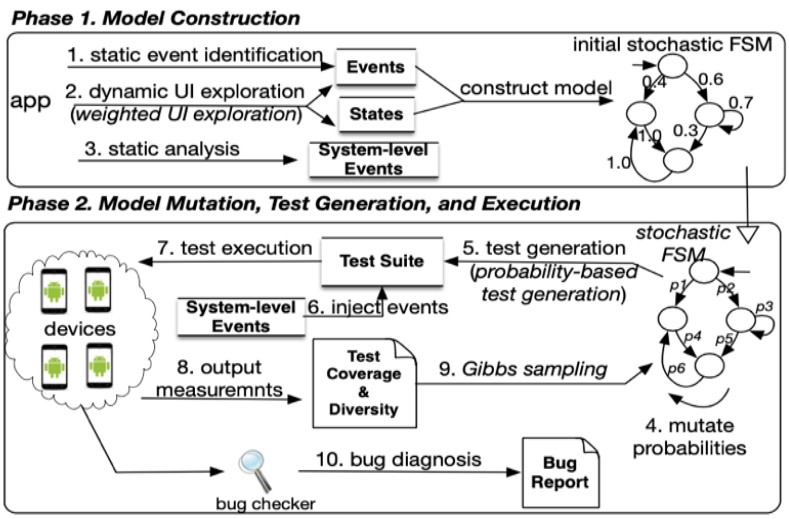
在模型构建阶段，Stoat 构造了一个随机有限状态机（FSM）来描述应用程序的行为。它使用动态分析技术，并且通过加权 UI 探索策略有效的探索应用行为。通过分析应用程序页面的 UI 层次结构推断输入事件，

并动态地优先执行它们，以最大化代码覆盖率。此外，为了识别一些潜在的丢失事件，执行一个静态分析来扫描应用程序代码中已注册的事件监听器。Stoat 在探索期间记录所有 UI 事件的执行频率，然后使用它们在模型中生成转换的初始概率值[3]。总体来说，在模型构建过程中，Stoat 推断可执行事件，并根据事件类型和执行频率对其执行进行优先级排序。

随后在模型突变、测试生成和执行阶段中，Stoat 利用 Gibbs 抽样来指导派生出测试用例的随机模型的变异，具有更好的目标值的模型将被接受为下一次迭代的突变和抽样。假设应用程序有 n 个应用状态($s_1, \cdots, s_n$)，每个状态 s_i($1 \leqslant i \leqslant n$)有 k 个事件转换（$e_1, \cdots, e_j, \cdots, e_k$)。Stoat 随机的决定是否改变每个应用状态 s_i 的转移概率。如果选择状态 s_i，则 Stoat 将 e_j 的原概率值 p_j 随机突变为一个新的概率值 p_j’，新生成的概率值 p_j’在 p_j 附近(它可以高于或低于 p_j)，因此新模型 M ’可以生成非常不同的事件序列 s。对于其余的转移概率，应用了类似的过程，但约束 p_1’ +…+p_j’ +…+p_k’ = 1 仍然成立。对于其他未选择的状态，它们的转移概率保持不变，以使新的模型 M ’通过突变的概率值有条件地依赖于之前的模型 M。这样在每次迭代中通过改变前模型 M 中的转移概率值，生成一个新的候选模型 M’，就完成了随机模型变异的过程。基于随机模型的不断变异，派生出来的测试用例将是多样化的，也就能够实现较高的代码覆盖率和模型覆盖率。

此外 Stoat 将系统级事件随机注入到 UI 级事件序列中，这种策略避免了在行为模型中包含系统级事件的复杂性，系统级事件可以揭示更多难以检测到的 bug。

总的来说，为了彻底测试一个应用程序，Stoat 会利用模型构建阶段生成的模型来迭代地指导测试生成，以获得高覆盖率和显示不同的事件序列。具体来说，Stoat 的执行是一个循环的过程。首先它会随机变异事件的转换概率，随机将随机将高概率事件变为低概率事件，将低概率事件变为高概率事件。然后根据转换后的概率生成测试，并在这个过程中随机的注入系统级事件。测试生成后执行测试，根据输出的度量（如代码覆盖率等）来判断测试的效果。根据测试的结果 Stoat 利用 Gibbs 抽样来决定新提出的模型是否应该被接受或拒绝，具有更好的目标值的模型将被接受为下一次迭代的突变和抽样；否则，它将以一定的概率被拒绝，以避免局部最优(如果被拒绝，原模型将被重用)。一旦检测到任何错误(如程序崩溃或者没有响应)，将使用相应的测试执行进一步的分析以诊断错误。具体流程图如下：



Stoat 已经被证实在 UI 探索和检测深度 bug 方面比其他技术更加有效，并且 Stoat 在测试中注入的系统级事件可以检测出很多意想不到的 bug。但 Stoat 也存在一些局限性。首先，在测试期间，Stoat 发出一个事件，等待它生效，然后发出下一个事件。这种同步确保了测试的完整性，但是它可能会错过那些只能通过快速操作来验证的 bug。其次，Stoat 可能会从模型中生成“不可行的”事件序列，这就导致测试无法正确执行。第三，Stoat 生成的模型仍然不完整，因为它不能捕获 UI 探索期间所有可能的行为，这仍然是 GUI 测试的一

个重要研究目标。

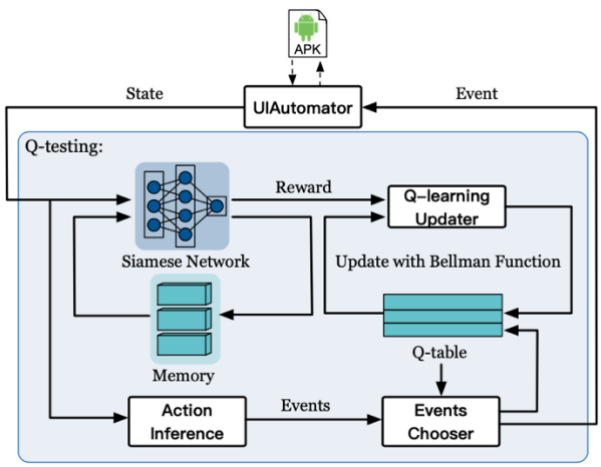
3 基于学习的自动化测试

基于强化学习的测试是在模型的基础上提出的，它采用激励机制。若在测试中某个操作探索到了未探索的空间，激励机制会给这个操作更高的激励值。测试会根据激励值的变化来不断的调整它的探索策略，这样测试就能涵盖到应用程序的更多功能，使得测试更加充分。基于强化学习的测试比基于随机和基于模型的自动化测试覆盖率更高，但它前期的训练和准备工作较为复杂，成本也相对较高，普适性不是很强。

3.1 Q测试

潘敏学等人提出了 q 测试[2]，这是一种基于强化学习的方法。q 测试以一种好奇心驱动的策略来探索 Android 应用程序，它将测试导向它所好奇的状态，从而高效地覆盖代码和发现 bug。q 测试利用内存集来记录之前访问过的部分状态，并指导测试不熟悉的功能。强化学习奖励是根据当前状态与内存集中记录的状态的差异来计算的。与以往的研究不同，好奇心驱动的探索是一种动态的适应策略。通过适应性，它可以发现状态重要性的变化，并将持续调整特定事件的奖励。另外，为了提高测试效率，q 测试中提出了一种神经网络，以功能场景的粒度划分不同的状态，将不同的场景绑定到不同的代码。q 测试使用这个模块来比较状态和计算奖励。在它的帮助下，q 测试将优先努力覆盖不同的功能，这有助于合理分配有限的测试时间，并将导致代码覆盖率在短时间内快速增长。

q 测试的工作流程类似于 MDP 的过程，q 测试在测试过程中与外部环境、待测应用程序(AUT)交互。在每个周期中，q 测试首先使用 UI 自动监控器（UIAutomator）观察应用程序的当前状态。然后将状态 St 与部分观测状态进行比较。这些状态存储在缓冲区中，其作用类似于 q 测试的内存，它能帮助 q 测试找出它不熟悉的状态，也就是好奇的状态。通过训练神经网络提取状态特征并进行比较。如果状态 St 与内存中的任何一个状态相似，则给出一个小的奖励。否则，该模块将给出一个较大的奖励，并将状态 St 添加到内存缓冲区中。奖励用于更新存在 q 表中的状态-动作对的值。所有按活动分组的状态-动作对的值存储在 q 表中，每次到达一个新的状态时，q 测试都会将相关的状态-动作对添加到 q 表中，并用一个较大的值初始化它，以鼓励新事件的执行。q 值更新后，q 测试将从 St 的 GUI 层次中推断出可执行事件，并参考 q 表选择一个动作 At。在大多数情况下，将选择具有最高值的动作。执行后，AUT 将响应 At，并开始另一个周期。工作流程图如下：



除了基于好奇心的策略之外，q 测试还利用了其他基于 Android 应用特性而设计的策略，包括特殊小部件的特殊处理和系统事件的注入。在 Android 应用程序中，RecyclerView 和 ListView 这两个部件能够以可滚动的方式显示视图集合。在大多数情况下，这些小部件将包含许多项，用户可以通过滚动查看新项。点击这些条目通常会导致类似的屏幕(具有相同的 GUI 层次结构和不同的内容)，并且只能触发相同的代码。若测试

时没有考虑到这些小部件，会浪费大量时间来测试不同的项。在 q 测试中，如果 RecyclerView 或 ListView 中的几个项目导致类似的状态，q 测试将忽略除第一个项目外的其他项目。考虑到 RecyclerView 和 ListView 在日常的应用程序中很常见，这个策略可以节省很多测试时间。另外，系统级事件的注入可以通过触发复杂的错误而使测试受益。与 Stoat 类似，q 测试将三种系统级事件纳入考虑范围，包括用户操作、广播消息和可从 Android 清单文件中提取的应用程序特定事件。q 测试在探索过程中随机注入并执行系统级事件。注意这些事件不包括在 q 表中，因为系统级事件的数量非常大，一旦在 q 表中添加了它们，q 测试可能会尝试在每个状态中执行它们，会大大增加 q 测试的时间和开销。

4 UI 兼容性测试

目前市场上存在不同的操作系统、不同的 Android 版本、不同的应用版本和多种多样的应用程序，当前已经实现的自动化生成测试输入的工具通常具有局限性，不能很好的适配各种设备，因此自动化测试中迫切需要加强的就是提高测试的兼容性、可重用性。具体来说，有三种常见的测试场景需要进行 UI 兼容性测试：（1）版本兼容性测试，开发者在不同的 Android API 上测试他们的应用程序。（2）设备兼容性测试，开发者在不同的 Android 设备上测试他们的应用程序。（3）第三方库测试，开发者使用其现有应用测试第三方库的新版本。

4.1 Mimic

为此，Taeyeon Ki 等人一个 Android 应用程序的自动化 UI 兼容性测试系统——Mimic[8]。Mimic 是专门为比较不同设备、不同 Android 版本和不同应用版本的应用的 UI 行为而设计的。Mimic 允许 Android 应用程序开发者轻松地对其应用程序进行向后和向前的兼容性测试，它还可以对稳定版本的应用程序和更新版本的应用程序进行清晰的比较。

Mimic 使用跟随-领导者模型的测试。多个设备可以并行使用，但一个设备会成为执行一系列 UI 操作的“领导者”，其他设备跟随领导者执行相同的 UI 动作序列。这个模型对于不同 Android 版本、设备类型和应用版本的 UI 兼容性测试非常有用。此外，Mimic 通过一个用于编写测试的简洁编程模型使测试变得容易。编程模型允许测试人员快速建立测试环境，并表达他们自己的测试逻辑。在执行测试逻辑之后，Mimic 会报告 UI 兼容性问题，比如使用不同的 UI 路径，显示不同的 UI 结构，抛出不同的异常，UI 性能的差异等等。Mimic 以两种形式提供测试结果。首先，它使用日志采集器捕获测试应用程序生成的所有日志。其次，Mimic 使用 Graph Generator 来显示测试期间发生的所有 UI 转换。这个 UI 转换图帮助测试人员可视化不同版本应用程序实例的行为。使用这些图，测试人员可以很容易地发现错误点，并比较不同版本之间的 UI 差异。

Mimic 的开发人员对从谷歌 Play 下载的几百个 Android 应用程序的评估显示，Mimic 可以有效地检测真实的错误，并报告不同 Android 或应用程序版本的 UI 的差异。得到了这些不同版本的 Android 或应用程序的 UI 差异后，在自动化测试自动生成测试输入时，就可以根据不同的版本生成具体不同的输入，对自动化测试的发展有着很大的帮助。

4.2 AppFlow

AppFlow[9]是一个综合高健壮性、高可重用性的 UI 测试系统，它由胡刚等人提出。AppFlow 利用机器学习来自动识别常见的屏幕和小组件，并且将不同的屏幕和小组件映射到规范的屏幕和小组件，这使得开发人员不必编写复杂、脆弱的逻辑代码来使用它们。规范的屏幕和小组件抽象了特定于应用的变化，使得应用之间共享测试变得很容易。另外 AppFlow 识别屏幕的能力能够使开发者专注于测试屏幕的特定流程，无需编写大量的样板代码，这就使得 AppFlow 的测试用例具有可重用性。

目前许多的应用程序属于同一类别，并实现类似的用户操作流（如京东、淘宝等一系列购物平台）。因此同一类别的应用之间在 UI 测试时，完全可以共享和重用部分测试。如果测试是作为一个整体指定的，那么测试就很难重用，因为不仅测试场景的实现不同，而且达到场景所需的步骤也不同。为此，AppFlow 将测试模块

化，当遇到新的同一类别的应用时，模块化测试可以从测试库中挑选并自动合成测试用例，以适应该应用的行为。

5 总结

在这篇文章中，我们详细的介绍了自动化测试的三大主流策略和具有代表性的自动化测试工具，这些测试工具都实现了较高的代码覆盖率和发现错误的概率。但是他们中的大多数都没有很好的解决测试的兼容性问题，这将是自动化测试输入生成工具未来研究和发展的主要方向之一。

References:

[1]. AppBrain Group. 2017. AppBrain. (2017). Retrieved 2017-2-18 from http://www. appbrain.com/stats/

[2]. Minxue Pan, An Huang, Guoxin Wang, Tian Zhang, and Xuandong Li. 2020. Reinforcement Learning Based Curiosity-Driven Testing of Android Appli- cations. In Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '20), July 18–22, 2020, Virtual Event, USA. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3395363.3397354

[3]. Ting Su, Jue Wang, and Zhendong Su. 2021. Benchmarking Automated GUI Testing for Android against Real-World Bugs. In Proceedings of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '21), August 23§28, 2021, Athens, Greece. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/ 3468264.3468620

[4]. Tianxiao Gu, Chengnian Sun, Xiaoxing Ma, Chun Cao, Chang Xu, Yuan Yao, Qirun Zhang, Jian Lu and Zhendong Su, Practical GUI Testing of Android Applications via Model Abstraction and Refinement. In 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)

[5]. Ting Su, Guozhu Meng, Yuting Chen, Ke Wu, Weiming Yang, Yao Yao, Geguang Pu, Yang Liu, and Zhendong Su. 2017. Guided, Stochastic Model- Based GUI Testing of Android Apps. In Proceedings of 2017 11th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, Paderborn, Germany, September 4–8, 2017 (ESEC/FSE'17), 12 pages. https://doi.org/10.1145/3106237.3106298

[6]. Ke Mao, Mark Harman and Yue Jia, Sapienz: Multi-objective Automated Testing for Android Applications

[7]. Peng Liu, Xiangyu Zhang, Marco Pistoia, Yunhui Zheng, Manoel Marques and Lingfei Zeng, Automatic Text Input Generation for Mobile Testing. In 2017 IEEE/ACM 39th International Conference on Software Engineering

[8]. Taeyeon Ki, Chang Min Park, Karthik Dantu, Steven Y. Ko, Lukasz Ziarek, Mimic: UI Compatibility Testing System for Android Apps. In 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)

[9]. Gang Hu, Linjie Zhu, and Junfeng Yang. 2018. AppFlow: Using Machine Learning to Synthesize Robust, Reusable UI Tests. In Proceedings of the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '18), November 4– 9, 2018, Lake Buena Vista, FL, USA. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3236024.3236055