基于模型进化的移动应用测试数据生成方法

杨森[1] 黄松[1] 惠战伟[1]

（陆军工程大学指挥控制工程学院 南京 2100000）[1]

摘 要 随着移动端互联网的快速发展，人们对移动应用的需求不断增强，如何有效的测试移动应用变得愈加重要。模型驱动策略下传统的状态机模型随着状态的增长，往往会出现状态爆炸的情况，而现有的移动应用的复杂度逐步攀升，极大的制约了该方法的实际应用，本文提出了一种基于模型进化的移动应用测试数据生成方法，能够有效地提升移动应用测试的效率。本方法的创新点在于：首先，基于模型进化，建立了一套全新的分析 UI 控件及行为特征的动态移动应用模型生成方法；其次，应用机器学习算法进化模型来指导测试用例生成，打破了之前随机生成测试用例的无序性，为测试用例的生成提供了新的指导；最后，建立了原型工具验证了本方法并选择了七种同类测试工具进行对比实验，结果表明，本原型工具的测试覆盖率较其他工具最优覆盖率平均提升了 5%~15%。

关键词 移动应用测试，模型驱动测试，模型进化，自动化测试

中图法分类号 TP311 **文献标识码** A

Test data generation method for mobile application based on model evolution

Yang Sen[1] Huang Song[1] Hui Zhanwei[1]

(Collage of command and control engineering, the Army Engineering University of PLA, 210000, Nanjing)[1]

Abstract With the rapid development of mobile Internet, people's demand for mobile applications is increasing, so how to test mobile applications effectively becomes more and more important. Model driven strategy under the traditional state machine model will explode as the growth of the states, but the complexity of the existing automatic applications gradually rises, which greatly restricts the practical using of this method. This paper proposes a mobile application test data generation method based on the evolution-model, which can effectively improve the efficiency of mobile application testing. The innovation of this method is as follows: first, based on model evolution, a new dynamic mobile application model is established to analyze the UI interface and behavior characteristics of mobile application. Second, the application of machine learning algorithms is used to evolve the model to guide the test cases generation, breaking the disorder of randomly generated test cases, which offers a new guidance for the generation of test cases. Finally, the prototype tool was established to verify this method and seven similar test tools were selected for comparison experiment. The results showed that the test coverage of this prototype tool was improved by 5%~15% on average compared with that of other tools.

Keywords Mobile application testing, Model-based testing, Model Evolution, Automated Testing

1 引言

据 CNNIC[1]发布的第 41 次《中国互联网络发展状况统计报告》，截止到 2017 年 12 月，中国网民规模已达 7.72 亿，互联网普及率 55.8%，手机网民规模 7.53 亿，占比高达 97.5%。随着移动端互联网的快速发展，人们对移动应用的需求不断增强，移动应用的数量成几何式增长，但是，如何确保他

们的质量仍是困扰开发人员的难题。移动应用不同于以往桌面应用的特点在于，首先，移动应用基于丰富的 GUI 控件的事件驱动实现其主体逻辑，其次，移动应用的开发生命周期短，更迭速度极快，手动测试代价大，速度慢，而现有自动化测试工具覆盖率低，效率不高。因此，如何有效的测试移动应用变得越来越重要。

当今，主流的移动应用测试方法[2]主要分为：一，随机探索策略，即随机生成测试输入序列，缺点是效率低下，主要代表是 Monkey[3] 和 Dynodroid[4]；二，系统探索策略。由于某些应用只在特定输入下出错，因此常常需要复杂的组合方法诸如符号执行或者进化算法进行测试，主要代表有 ACTEve[5]，EvoDroid[6]等；三，模型驱动策略，即通过先建立待测应用 GUI 模型（常见模型有限状态机模型 FSM），然后依据模型按照一定策略遍历，最终系统地测试被测应用的所有状态，主要代表由 GUIRipper[7]，A^3E[8]，PUMA[9]等。

对于模型驱动策略，因为状态机模型随着模型状态的增长，会出现状态爆炸的情况，而现有的自动应用的复杂度逐步攀升，极大的制约了该方法的实际应用，因此现有的方法常是以状态转化覆盖为目标从建立的模型中生成随机测试用例，这往往导致测试的不充分，原因是显而易见的，首先，初始建立的模型往往并不能准确的表达原被测应用，根据最新的一份移动应用自动化测试研究表明，当前的模型驱动策略分析工具通过静态地分析移动应用分析 UI 控件的平均代码覆盖率不到 25%[2]，仅接近随机探索策略工具代码覆盖率的一半，其次，从建立的模型中随机生成测试用例的方式缺乏效率与指导。针对以上问题，本文提出了一种基于模型进化的动态移动应用测试数据生成方法。这种方法有以下两个优点：一，基于模型进化，建立动态可变模型分析搜索移动应用的控件及行为特征，代码覆盖率较同类工具提升了 5%~15%，提升效果明显；二，应用机器学习算法进化模型来指导测试用例生成，打破了之前随机生成测试用例的无序性，为测试用例的生成提供了新的指导，最后，依据此方法建立原型工具在实际应用上进行测试，达到了良好的效果。

本文第二节将具体介绍移动应用测试领域相关工作，第三节将详细阐述基于模型进化的移动应用测试方法，第四节介绍开发的原型工具及实例演示，最后是对本方法的总结以及对下一步工作的展望。

2.相关工作

2.1 Android 平台

Android 系统[10]架构由四部分组成,最下层是 Linux 内核,这一层提供最底层的系统服务,例如内存管理、网络通信、进程管理等。第二层是库和运行环境,同样是为应用层提供服务,这里包含了一些 C/C++库、SQLite 数据库、绘画引擎和 Java 核心库等, 还有最关键的 Dalvik 虚拟机。第五层是应用程序框架层,这一层提供了大量的 API 供开发者调用。系统架构如图一所示。

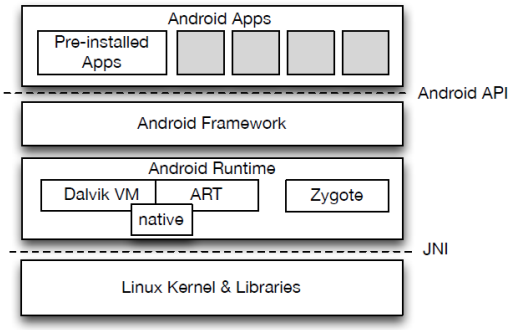


图 1：Android 系统架构

Android 应用程序由组件组成,Android 系统共有四大基本组件[11]:Activity、Service、Broadcast Receiver and Intent、Content Provider。Activity 直接负责人机交互,是应用可视化界面的载体，其负责

接受用户输入，在移动测试中最重要的一环；Service 在后台运行没有可视化界面,通常作为辅助作用出现,例如在后台提供服务、监控其他组件等；Broadcast Receiver and Intent 实现进程间相互通信，常是触发应用状态改变的隐形因素；Content Provider 则用来给不同应用提供内容访问。

2.2 模型驱动测试

模型驱动测试由来已久，其在传统的软件工程领域有着一套完整地解决方法[12]。 Dala[13]定义了基于模型测试的三个要素：描述程序行为的模型，测试用例生成算法以及测试用例执行工具，而如何构建模型则是最为关键的要素。 T.Takala[14]最早研究了状态机模型在移动应用功能测试领域的应用可行性。其研究的核心是讨论能否将传统的 GUI 建模方法直接应用到移动应用领域,实验结果最终证实为移动应用建模及基于模型的测试方法是可行的，但是现有工具都效率太低。自此，有许多静态模型测试工具先后提出。

GUIRipper[7]从初始状态开始逐步建立模型，每当访问一个新的状态，其保存下该状态下所有可以产生的事件，然后系统的访问这些事件直到检测不到新的状态产生，然后重启移动应用开始下一次测试，根本上是实现了一种基于深度优先搜索的测试覆盖。ORBIT[15]使用了和前者同样的策略，不过加入了对移动应用源码的静态分析，通过分析移动应用 GUI 控件与应用状态之间的关系，进一步提高了测试的效率。SwiftHand[16]则通过最少化应用重启次数进一步提高了测试的效率，但是在测试中其并没有考虑到系统事件的影响。PUMA[9]则是一款移动应用框架，可以根据实际需要调整测试策略。

使用模型驱动策略测试移动应用，可以极大地减少随机测试策略产生的冗余测试数据，但是现有的测试工具都是基于 GUI 界面定义有限状态机的状态，但是，往往许多事件并不会改变移动应用的 GUI 界面，但的确引起了应用状态的改变。这种情况下，之前建立的模型就会失效，进而丢失掉对某些控件的测试。

3.基于模型进化的数据生成

3.1 初始模型构建

3.1.1 动态界面控件分析

因为安卓应用的 GUI 界面其实是一个分层结构，由前端显示的 UI 控件及布局页面嵌套而成，对于每一个 GUI 页面来讲，其根本上来讲是一棵树，非叶节点代表布局部件，叶节点代表可点击 UI 部件。如图 2 所示，图 2-1 所代表的 GUI 界面实际是由图 2-2 定义的，整个页面包含在一个 Framlayout 布局里，其中又包括一个 LinearLayout 布局和 View 控件，分别代表上半部分功能界面以及下半部分页面导航栏。

在安卓平台上，整个页面布局可以通过 hierarchyViewer 和 uiAutomator[17]工具获得，这里使用了 uiAutomator 来获取页面信息，通过分析页面中的控件类型，就可以分析出触发下一个界面的事件（如，点击，编辑等）。

对于移动应用来说，其总是存在一个初始页面，那么从初始页面开始，将每一个到达的 GUI 页面定义为一个状态，而每个页面之间的转移由一个输入事件触发，当移动应用退出或者崩溃时，其最后一个所处的状态即为最终状态。按照这种方法就可以构建出 GUI 页面间的动态转移图。

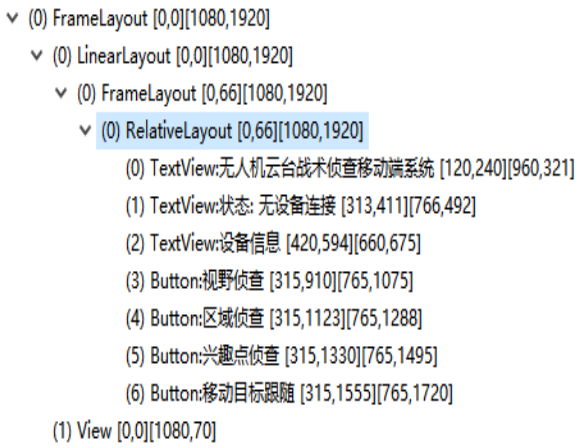
用一个四元组 M 来定义状态模型，这种模型能够完整描述 GUI 控件间的转化关系，定义如下：

$$M = (S, S_0, I, Y)$$

其中：S代表状态集合

$S_0 \epsilon S$代表初始状态

(1)移动应用页面截图



(2)获取到的页面分层树

图 2：GUI 布局示例

I代表输入事件表

Υ代表$S \times I \to P(S)$表明在状态s下所能到达的状态及概率的集合。比如s状态有s_1，$s_2 \ldots s_n$ 共 n 个可达后即状态，每个状态转换间有对应的触发事件e_i，定义其转化概率为p_i，即

$$P(s) = \{(s, s_1, e_i, p_i) | 1 \le i \le n, \sum_1^n p_i = 1\}$$

3.1.2 静态事件流分析

前面提到了通过动态分析页面的布局文件来查找页面间的触发事件，但是存在一些触发事件是通过布局文件找不到的，比如滑动操作，长按操作，这就需要通过静态分析移动应用的源文件来查找每个控件相关的触发事件对 3.1.1 中的模型进行完善。

此外，仍需要对建立的模型进行压缩[18]，因为模型的某些 GUI 界面虽然发生了改变，但是实质状态并没有变化。首先，当页面存在 CheckBox，TextView 等输入控件时，由于其仅引起细微的变化，故不予考虑产生新的状态，其次，当存在 ListView 嵌套内容控件时，仅区分有内容和没有内容两种状态。

3.1.3 定义初始状态转化概率

由于移动应用测试中，每个状态存在多个触发事件引起下一个状态，当构建测试序列时，就需要根据这些控件的特点构建评价模型，优先选择覆盖率提高更大的触发事件，进而最大的提升测试覆盖的效率。

从三个角度考虑触发事件的优先级，首先，如果该事件触发的下一个页面含有更多的可触发事件，那么该事件有较大的概率覆盖更多控件；第二，不同的控件不同的触发事件被选择的概率也不同，统计了 Rico 数据集[19]中每类 App 的每种控件可达子控件的个数，依据此来赋予 UI 权值指标，见表 1；第三，将模型动态建模中每个事件记录下来，执行过的事件下次被选择的机会应该减少，保证建立初态模型过程中能更加均匀的访问更多的控件。

表 1 UI 控件权值表

控件类型	权值
菜单项（menu）	2
下拉框（Spinner）	1.5
输入栏（EditText）	0.8
滑动/拖动	0.5
切换类（Switcher）	0.3
其他	1

综上，定义了如下所示的单个触发事件权值评价模型：

$$p_{e_i} = \alpha * U_{e_i}/\bar{U} + \beta * T_{e_i} - \gamma * R_{e_i}/\bar{R}$$

其中，

α, β, γ 代表经验参数，依次设为 0.5，0.5，1

U_{e_i}代表事件e_i触发的下一状态页面内 UI 控件个数

$\bar{U}$代表页面平均控件个数

T_{e_i}代表控件类型

R_{e_i}代表当前触发事件总执行次数

$\bar{R}$代表该页面控件平均触发次数

在初态模型构建完成时，计算单个状态页面所对应的触发事件计算状态转化概率，及归一化当前页面所有触发事件的权值。

$$\bar{p}_{e_i} = p_{e_i}/p_e$$

其中：

p_{e_i}为当前事件权值

p_e为当前页面所有触发事件权值之和

3.1.4 *初态模型构建算法*

输入一个移动应用，先动态分析其 UI 控件组成得到每个页面状态对应的触发事件表，然后静态分析源代码对上述事件表进行扩充完善，最后再为每个状态下所有转化事件定义状态转化概率，最终构建其对应的初态模型为后续模型进化提供初始输入，详细流程见算法一。

算法一 初态模型构造算法

函数：getInitialModel（A）

输入：待测移动应用 A

输出：初态模型 M

```
1  S0=getStart(A)  //获取初始状态
2  U=getEvents(startActivity)  //获取初始页面可以触发下一状态的 UI 控件，包括 3.1.1 的动态分析和 3.1.2 的静态分析
3  S={S0}  //初始化状态集合
4  While(U！=null&&！timeout)
5    foreach(e in U)
6      updateModel(M,e, S0, pi)
7      //更新模型中S0的状态转化关系与事件触发权值
8    end
9    em = getMax(p)
10   sn=execute(em)  //优先执行权值较大的触发事件
11   S=S ∪ { sn}
12   updateModel(M,S0 , sn,em)
13   M=getNextStateModel(M, sn)
14   //探索新状态之下的转化模型
15 end
16 S=getAllState(M)  //获取所有状态
17 foreach(s in S)
18   E=getAllEvent(s)
19   updateModelPossibility(M,s,e,p̄)
20   //为每个状态按照 3.1.3 中模型更新初始转换概率
21 end
22 return(M)
```

3.2 基于模型进化算法的测试数据生成

3.2.1 *测试覆盖率计算*

根据移动应用测试覆盖的特点，选取了两个指标作为测试覆盖率计算的考虑因素：活动（Activity）覆盖[20]，代码（Code）覆盖[21]。选取这两个因素是因为，首先，活动是移动应用 GUI 界面的来源，每一个活动都是一种状态，如果漏测试一个状态，就会丢失对该活动中更多 UI 控件的测试；其次，代码覆盖是最基础的覆盖方法，这里需要指出的是，对于开源移动应用，代码覆盖是指对其进行分支覆盖，对于不开源移动应用，代码覆盖时对其进行方法覆盖。由此，定义模型覆盖度评价模型：

$$\mathrm{V}(L_M) = \mathrm{a} * \mathrm{AR}/\mathrm{AT} + \mathrm{b} * \mathrm{CR}/\mathrm{CT}$$

其中：

a, b 代表经验参数，设为 0.7，0.3

AR 代表到达的活动个数

AT 代表总活动个数

CR 代表覆盖到的代码行数

CT 代表总代码数

3.2.2 生成测试数据

对于一个给定的模型，采用概率最大化算法去寻找其测试序列。即从模型的初始状态开始，每次都选择概率值最大的触发事件触发下一状态，这样生成一条概率最大化的测试路径，就得到当前模型下最高效的测试覆盖。

将每一次测试序列定义为$\mathrm{L}_M = (e_1, e_2 \dots e_n)$，其中$e_1$代表从模型 M 的初态$s_0$开始进入下一状态$s_1$的触发事件，$e_2$代表从状态$s_1$进入下一状态的触发事件，以此类推，$e_n$代表进入结束状态前最后一个触发事件。

3.2.3 吉布斯采样算法

吉布斯算法[22,23]源自马尔可夫链蒙特卡罗方法[24]。该方法的基本思想是：对于一个给定的概率分 $P(X)$，若要得到其样本，可以构造一个转移矩阵为 P 的马尔可夫链，使得该马尔可夫链的平稳分布为 P(X)，这样，无论其初始状态为何值，假设记为x_0，那么随着马尔科夫过程的转移，得到了一系列的状态值，如：$x_0, x_1, x_2, \cdots, \mathrm{x}_n, \mathrm{x}_{n+1}, \cdots,$，如果这个马尔可夫过程在第 n 步时已经收敛，那么分布 $P(X)$ 的样本即为$\mathrm{x}_n, \mathrm{x}_{n+1}, \cdots$。

吉布斯算法假设需要从目标概率密度函数$\mathrm{p}(\theta)$中进行采样，即根据马尔可夫链去随机生成一个序列：$\theta(1) \to \theta(2) \to \cdots \theta(\mathrm{t}) \to \cdots$，其中，$\theta(\mathrm{t})$表示的是马尔可夫链在第 t 代时的状态。

采样算法的过程：首先初始化状态值$\theta(1)$，然后利用一个已知的分布$\mathrm{q}(\theta \mid \theta(\mathrm{t}-1))$生成一个新的候选状态$\theta(*)$，随后根据一定的概率选择接受这个新值，或者拒绝这个新值，在吉布斯采样算法中，这个概率为：

$$\alpha = \min\left(1, \frac{\mathrm{p}(\theta^*)}{\mathrm{p}(\theta^{t-1})}\right)$$

这样的过程一直持续到采样过程的收敛，当收敛以后，样本$\theta(t)$即为目标分布 $\mathrm{p}(\theta)$中的样本。

3.2.4 模型进化算法

按照 3.2.3 算法的一般方法，设定模型进化的目标函数

$$\mathrm{p}(\mathrm{M}) = \frac{1}{Z} * e^{-a*V(S_M)}$$

其中：

Z 代表归一化分布函数

a 代表常数变量

$V(S_M)$代表状态 M 下生成的测试序列的实际测试覆盖率。

则其对应的接受概率为

$$\alpha = \min(1, e^{-a*(V(S_M)-V(S_N))})$$

其中：

$V(S_M)$代表原模型测试覆盖

$V(S_N)$代表演化后模型的测试覆盖。

将 3.1 定义的模型的初始状态M_0作为吉布斯算法中的初始状态值$\theta(1)$，随机改变M_0中每个状态触发事件的转化概率，按照吉布斯算法的接受概率选择是否接受新的待选模型，最终，使得模型向着覆盖率最大的方向进化，达到最大化覆盖率的目标，具体流程见算法二。

算法二 模型进化算法

函数：evolveModel（M_0）

输入：初态模型M_0

输出：演化模型 M

```
1  S0=getStates(M0)   //获取模型M0的所有状态
2  foreach(s in S0)   //随机调整每个状态触发事件的转化概率
3     E= getEvents(M0)
4     If(Random(0,1)>0.5)
5        E′=randomAdjust(E)   //调整转化概率得到新的演化概率，这里的调整选取了随机的微小变化
6        M=update(E′)
7  end
8  L=getTestSuit(M)
```

9 //获取新模型下的概率最大化测试序列

10 execute(L)

11 if $\alpha(M_0, M)$>random(0,1)//按照吉布斯采样接受率选择是否接受新模型

12 return M

return M_0

4 实验验证

4.1 原型工具设计

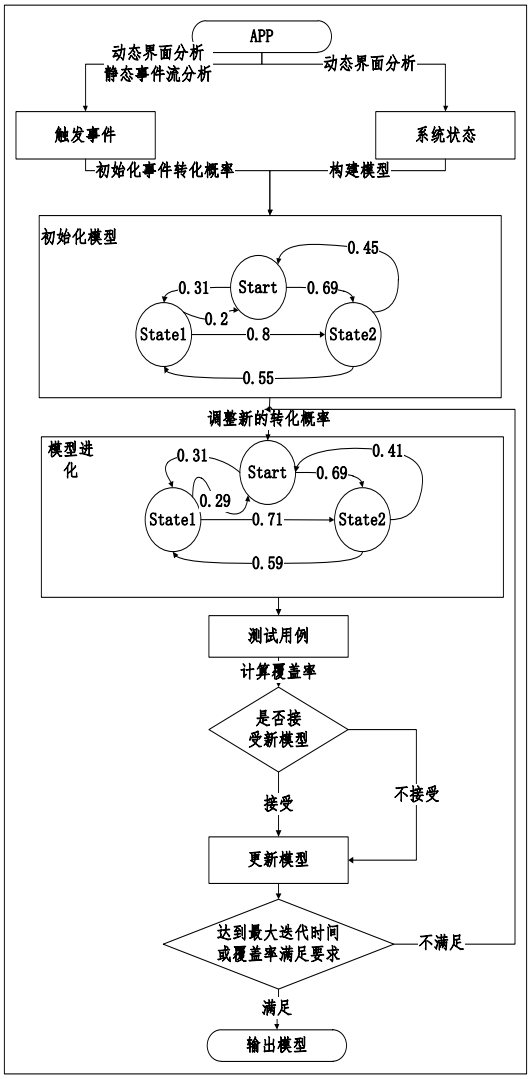


图 3：原型系统实现逻辑流程图

针对所提出的方法，实现了原型工具并完成了相关的实验，该原型工具使用了 A^3E[8]动态分析界面控件，使用了 Soot[25]静态分析潜在触发事件，在生成测试用例时，从当前模型中生成了前 50 条概率最大测试序列，每条序列最多包含 30 个触发事件，采用 Android uiautomator 以及 Android ADB 工具执行测试用例并采用 Emma[26]测试开源移动应用的分支覆盖率，用 Ella[27]测试闭源移动应用的方法覆盖，具体的逻辑流程见图 3。

试验平台运行在 Ubuntu 14.04 系统上，为其分配了 14G 内存，以及 4 核 2.50GHz 的 CPU，安装了 Android SDK 18 和 19 版本，试验中使用安卓虚拟机运行测试用例，为每个虚拟机分配了 2G 内存，每次测试采用了 2 小时进行测试，并计算最终覆盖率。

4.2 验证实验

4.2.1 实验设计

测试了一款移动应用 Capture，Capture 是笔者开发的一款控制无人机实时回传图像的应用，该应用旨在以现有无人机平台为基础，实现战术背景下的区域空中侦察任务。通过测试该程序，可以分析测试过程中的模型建立，吉布斯取样，测试用例执行，测试覆盖率计算等过程，具体阐述本方法的实现过程。

该应用主要实现了视野侦查、区域侦查、兴趣点侦查、移动目标跟随等四大功能，每一个按钮分别对应了其一个主要分支子功能，主要功能页面如图 4 所示。

4.2.2 测试数据生成

按照静态分析和动态分析的方法，发现该应用存在 48 个状态，并且，根据模型间触发事件的转化概率，构建如图 5 所示的状态机模型。图中 Sn-pm 代表状态 n 下第 m 个转化事件的转化概率。从主入口进入的触发事件转化概率如表 3 所示。

模型进化时，随机调整每个状态触发事件的转化概率，这里的调整量选取了相对微小的变化，防止模型进化的步长过大导致实验失败。表 4 给出了

第一次进化后模型的变化情况，由图可以看出，触发区域侦察事件的转化概率进一步提升，这是因

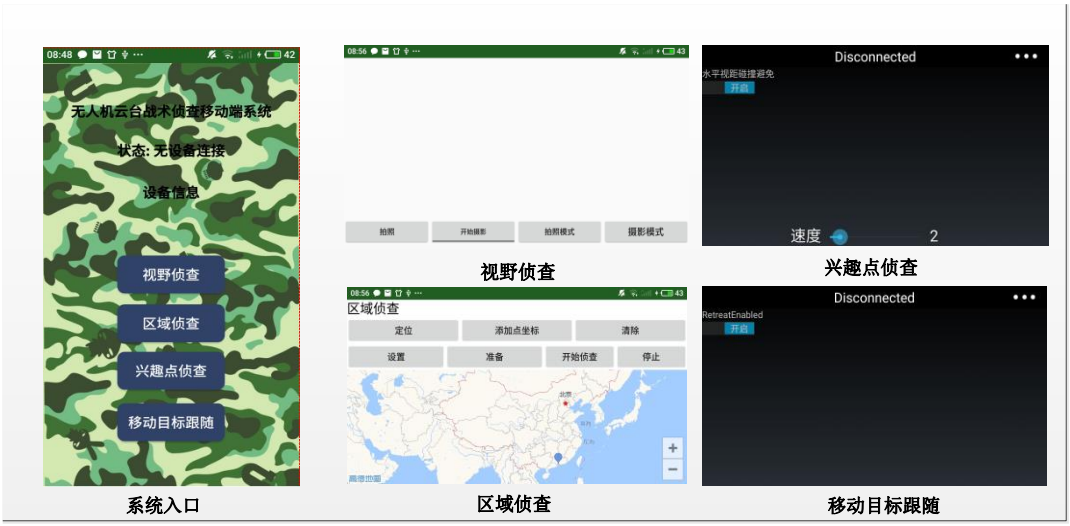


图 4：无人机侦察系统实例

表 3 入口状态触发事件初始转化概率

触发事件	resource-id	转化概率
视野侦查	com.dji.FPVDe:id/btn_open	0.10
区域侦查	com.dji.FPVDe:id/btn_open2	0.51
兴趣侦查	com.dji.FPVDe:id/btn_open3	0.19
移动目标跟随	com.dji.FPVDe:id/btn_open4	0.20

表 4 入口状态触发事件模型进化后的转化概率

触发事件	resource-id	转化概率
视野侦查	com.dji.FPVDe:id/btn_open	0.12
区域侦查	com.dji.FPVDe:id/btn_open2	0.41
兴趣侦查	com.dji.FPVDe:id/btn_open3	0.25
移动目标跟随	com.dji.FPVDe:id/btn_open4	0.19

为该功能的子功能模块较多，代码量也最多，因此测试资源分配时也应该进一步向这一测试分支转移，这符合当初设计这款应用时的实际情况，因为该功能是唯一需要调用第三方接口库的功能，而且实现区域侦察子功能还需要调节更多参数，因此其代码量是其他功能的两倍多，因此，为了充分测试该应用，理应增大该功能的转化概率。而其他三个子功能，特别是移动目标跟随以及兴趣侦查，实现逻辑基本相同，因此进化后的模型为其分配几乎相同的转化概率是符合预期的。综上所述，模型进化过程符合预期，并且能够为提高测试效率带来启发。

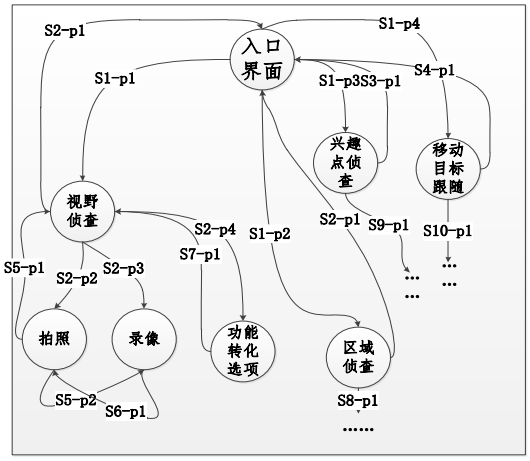


图 5 原型系统状态机模型

4.2.3 结果分析

本次测试，模型共进行了 25 次进化迭代，对比第一次生成的模型以及最终模型以及测试结果，可以得知，转化概率不断朝着能使有着更大测试代码覆盖的分支倾斜，推动测试的效率的提高。

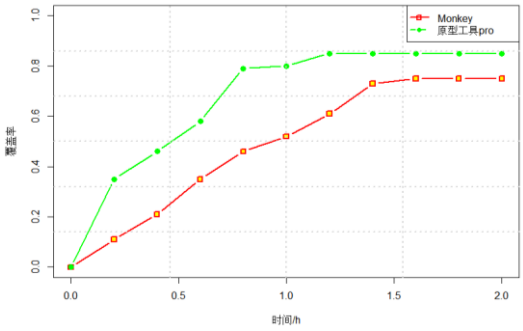


图 6 模型覆盖率增长曲线

图 6 给出了本次测试中模型覆盖率随时间变化的图像，由图可知，由于原型工具有启发信息，测试过程的覆盖率增长速度明显高于另一工具，说明本工具确实提高了测试的效率。表 5 对比了本工具与 Monkey[3]工具在两小时测试时间内测试覆盖度上的对比。由表可以看出，本工具在测试覆盖率上提升了 10%~20%。

表 5 最终测试覆盖率结果

工具	活动覆盖率	代码覆盖率
Monkey	80%	39%
原型工具	86%	48%

4.3 对比实验

4.3.1 实验设计

依据公开的标准开源 APP 数据集[2]，选取了 68 个开源 APP 作为本次对比实验的基准数据集。这个数据集中有 52 个源于 Dynodroid[4]的验证，3 个源于 GUIRipper[7]的验证，5 个源于 ACTEve[5]的验证，10 个源于 SwiftHand[16]的验证。

同时，为了对比本方法效率的提升，选取了七个开源的研究原型工具来进行比较，首先，Monkey 测试工具；其次，ACTEve[5]，一款基于系统开发策略的测试工具；第三，Dynodroid[4]，基于随机策略的测试覆盖工具；第四，A^3E[8],一款系统化 UI 搜索工具；第五，GUIRipper[7]，一款基于深度优先搜索的测试覆盖工具；第六，PUMA[9]，一款基于路径覆盖的测试工具；第七，SwiftHand[16]，一款通过最少化应用重启次数进一步提高了测试的效率的测试工具。

4.3.2 实验分析

表 6 给出了八款测试原型工具在 68 个公开的开源 App 数据集[2]的测试覆盖率上的结果。其中，第一列给出了 App 名，第二列给出了被测 App 的版本号，Dyn 代表 Dynodroid[4]方法的测试覆盖率，GuiR 代表 GUIRipper[7]方法的测试覆盖率，SwiftH 代表 SwiftHand[16]方法的测试覆盖率，Pro 代表本方法的原型工具的代码覆盖率，覆盖率结果中，X 代表测试中被测件发生了崩溃、闪退等异常问题。由表可知，首先，在 39 个被测 App 上，本原型工具的测试覆盖率优于其他工具，其次，总体来看，本原型工具的测试覆盖率较其他工具最优覆盖率平均提升了 5%~15%。

表 6 代码覆盖率结果

APP	版本	Monkey	Acteve	Dyn	A3E	GuiR	PUMA	SwiftH	Pro
Amazed	2.0.2	X	50.58%	64.04%	54.08%	X	49.42%	10.12%	81.12%
AnyCut	0.5	64.19%	14.86%	67.61%	11.09%	9.53%	56.22%	1.44%	79.23%
Divide&Conque	1.4	85.44%	50.59%	81.69%	44.33%	X	46.48%	0.00%	88.12%
LolcatBuilder	2	18.93%	24.09%	25.00%	22.71%	25.53%	14.75%	7.26%	22.12%
MunchLife	1.4.2	70.15%	41.20%	71.32%	42.32%	45.93%	45.90%	11.28%	65.32%
PasswordMaker	1.1.7	51.74%	27.84%	46.65%	27.67%	32.59%	29.35%	10.65%	61.92%
Photostream	1.1	28.72%	25.39%	24.80%	9.05%	9.76%	24.14%	4.17%	26.11%
QuickSettings	1.9.9	54.22%	27.52%	35.68%	22.44%	24.19%	21.91%	0.00%	43.12%
RandomMP	1	78.59%	47.95%	81.66%	14.46%	40.42%	48.16%	1.20%	82.32%
SpriteText	-	57.84%	57.14%	58.68%	60.47%	58.79%	59.28%	0.00%	58.77%
SyncMyPix	0.15	1.61%	6.41%	21.16%	5.35%	8.12%	18.47%	2.21%	23.46%
Triangle	-	68.02%	65.74%	83.03%	76.73%	66.24%	73.28%	0.00%	75.30%
A2DP Volume	2.8.1	47.12%	25.77%	37.82%	4.92%	16.87%	19.54%	0.00%	48.32%
aLogCat	2.6.1	67.70%	42.22%	69.00%	37.06%	38.84%	61.52%	0.00%	73.33%
AardDictionary	1.4.1	59.75%	16.06%	46.00%	11.64%	11.94%	13.21%	0.00%	61.12%
BaterryDog	0.1.1	64.84%	54.98%	65.16%	13.28%	12.35%	18.26%	4.82%	60.09%
FTP Server	2.2	14.53%	12.95%	15.22%	1.93%	6.16%	14.28%	0.00%	9.88%
Bites	1.3	46.57%	18.43%	32.72%	7.16%	15.28%	15.75%	0.00%	50.18%
Battery Circle	1.81	70.88%	50.00%	81.77%	47.58%	61.06%	42.18%	32.10%	81.23%
Addi	1.98	16.61%	16.74%	18.32%	16.85%	13.18%	16.84%	0.00%	13.13%
Alarm Clock	1.7	0.90%	45.49%	70.02%	17.39%	36.50%	17.39%	0.00%	73.23%
Manpages	1.51	68.59%	57.55%	71.70%	49.64%	49.64%	58.23%	0.00%	66.35%
Auto Answer	1.5	31.17%	29.96%	47.15%	28.11%	30.24%	31.17%	11.88%	35.25%
HNDroid	0.2.1	41.18%	17.47%	52.17%	29.07%	27.03%	15.49%	0.00%	38.41%
Multi SMS	2.3	66.01%	26.03%	75.44%	16.78%	11.70%	24.16%	0.00%	81.21%
World Clock	0.6	83.78%	81.00%	93.89%	84.11%	82.46%	84.44%	0.00%	97.24%
Nectroid	1.2.4	34.61%	38.15%	67.27%	28.29%	22.25%	57.37%	37.04%	63.29%
aCal	1.6	23.56%	8.02%	16.64%	8.00%	13.43%	12.38%	0.00%	21.28%
Jamendo	1.0.6	53.43%	3.47%	3.88%	13.09%	16.53%	9.63%	2.52%	69.63%
Androidomatic	1	80.81%	X	82.69%	18.26%	14.72%	29.32%	10.63%	63.69%
Yahtzee	1	47.54%	11.44%	55.27%	7.01%	31.29%	8.18%	0.00%	59.98%
aagtl	1.0.3	15.13%	11.98%	28.11%	0.37%	14.56%	13.35%	0.00%	31.23%
Mirrored	0.2.3	73.37%	44.12%	66.63%	32.29%	43.24%	30.16%	28.43%	59.69%
Dialer2	2.9	39.26%	31.85%	53.46%	29.44%	24.75%	30.52%	0.00%	77.61%
FileExplorer	1	45.14%	45.14%	58.87%	45.14%	45.14%	32.06%	23.04%	61.13%
Gestures	1	48.15%	38.24%	52.63%	38.24%	31.62%	23.58%	0.00%	46.68%
HotDeath	1.0.7	69.20%	52.22%	54.57%	3.22%	37.08%	44.85%	0.00%	60.09%
ADSdroid	1.2	X	34.51%	77.43%	67.95%	53.73%	23.60%	0.32%	66.89%
myLock	42	30.50%	32.47%	35.02%	10.52%	33.62%	7.30%	0.00%	39.98%
LockPG	2	73.68%	62.41%	78.74%	54.55%	38.17%	33.08%	26.53%	75.58%
aGrep	0.2.1	X	X	57.35%	X	X	X	10.28%	55.68%
K-9Mail	3.51	4.61%	2.87%	5.11%	2.88%	4.57%	2.74%	0.00%	7.33%
NetCounter	0.14.	70.65%	35.58%	62.87%	36.19%	33.69%	34.53%	0.00%	73.28%
Bomber	1	72.61%	62.25%	71.43%	66.33%	54.69%	48.00%	0.00%	74.45%
FrozenBubble	1.12	X	49.09%	68.49%	X	X	X	0.00%	73.79%
AnyMemo_135	8.3.1	22.33%	X	21.73%	9.45%	3.99%	X	0.00%	25.15%
Blokish	2	41.44%	40.48%	49.92%	35.75%	39.85%	32.60%	20.10%	52.64%
ZooBorns	1.4.4	67.08%	52.78%	69.83%	55.01%	49.31%	18.74%	0.00%	43.68%
ImportContacts	1.1	76.74%	3.68%	74.37%	3.73%	5.63%	28.28%	2.34%	79.93%
Wikipedia	1.2.1	41.56%	X	38.79%	25.42%	22.44%	5.91%	0.00%	33.98%
KeePassDroid	1.9.8	9.82%	4.49%	14.43%	4.39%	1.30%	4.55%	0.66%	12.14%
SoundBoard	1	47.58%	47.58%	63.64%	47.58%	42.28%	32.43%	0.00%	78.10%
CountdownTim	1.1.0	61.67%	65.42%	77.07%	43.64%	47.22%	42.45%	23.97%	81.23%
Ringdroid	2.6	X	X	X	X	X	11.31%	0.00%	X
SpriteMethodTe	1	70.19%	63.96%	38.36%	6.35%	6.06%	3.86%	3.70%	81.23%
Translate	1.6	50.19%	30.39%	50.13%	31.61%	30.55%	29.76%	0.00%	51.25%
BookCatalogue	3.8	30.64%	4.28%	9.85%	4.40%	5.44%	3.38%	0.00%	28.85%
TomdroidNotes	2.0a	55.80%	34.48%	46.61%	4.24%	17.37%	20.35%	0.00%	57.73%
Wordpress_394	0.5.0	6.90%	4.62%	6.99%	0.54%	3.48%	4.26%	0.00%	8.33%
Mileage	3.1.1	41.53%	24.39%	31.01%	2.97%	18.68%	21.06%	3.97%	39.98%
Sanity	2.11	28.66%	6.12%	1.13%	8.88%	9.52%	9.84%	0.00%	31.23%
DalvikExplorer	3.4	X	43.74%	43.81%	29.23%	5.40%	46.39%	5.39%	51.52%
MiniNoteViewe	0.4	46.15%	12.84%	33.48%	16.37%	8.39%	31.29%	0.00%	45.78%
MyExpenses	1.6.0	56.14%	14.49%	35.01%	21.95%	9.99%	26.84%	0.00%	61.19%
LearnMN	1.2	55.08%	39.07%	73.19%	11.05%	26.83%	6.68%	2.22%	61.28%
TippyTipper	1.1.3	82.65%	29.19%	53.70%	45.85%	31.35%	43.09%	14.52%	89.91%
WeightChart	1.0.4	52.35%	23.37%	56.85%	3.80%	17.73%	16.06%	0.00%	69.98%
WhoHasMyStuf	1.0.7	76.21%	35.56%	72.44%	44.26%	33.46%	55.63%	10.15%	84.41%

图 7 给出了不同工具间代码覆盖率的统计箱线图，图中的凹槽对比了各工具在统计意义上是否有显著性差异，即若两个箱的凹槽互不重叠，则表明它们的中位数有显著差异。由图可知，本原型工具在测试集上的表现优于其他工具，测试结果也与其他工具有显著性差异，此外，由于 GUIRipper[7]，A^{3}E[8]，PUMA[9]都采用了路径覆盖策略，所以测试结果的差异并不明显。

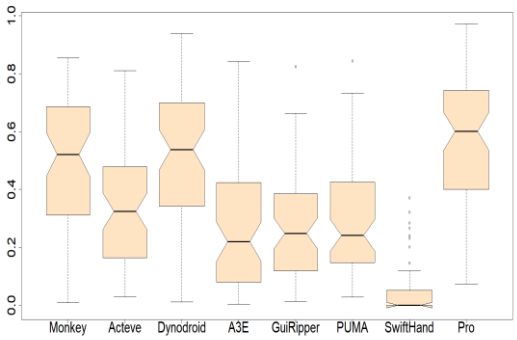


图 7 代码覆盖率箱线图

4.3.3 结果分析

通过实验，对比了各原型工具的测试效率，首先，随机测试策略测试工具，如 Monkey[3]和 Dynodroid[4]等，这些工具的测试完全随机，对于某些设计复杂的移动应用，测试效率降幅明显，这是由于其测试的随机性赋予了每种可达子状态均等的触发概率，而事实上每种可达状态由于包含子控件个数的不同，其访问的价值也应当不同，否则就会造成有限访问资源（如时间等）的浪费，这也是原型工具为不同的 UI 控件赋予不同权重的初衷，实验结果也表明本方法相对随机策略测试工具效率有了 10%的提升。

第二，系统探索策略测试工具，如 ACTEve[5]，SwiftHand[16]等，这些工具的设计初衷并不在于提高代码覆盖率，比如 ACTEve[5]旨在寻找被测试应用中的错误，SwiftHand[16]则重在尽可能减少被测应用在测试过程的重启次数，所以这些方法在代码覆盖率上的表现并不是很好。

第三，模型驱动策略，如 GUIRipper[7]，A^{3}E[8]，PUMA[9]等，这种工具采用了深度优先搜索遍历控件，这种方法造成了大量控件的重复访问，严重地制约了测试效率，而本方法通过不断进化模型，每次进入新的迭代时，优先选择了覆盖率提升最多的测试用例，因此测试覆盖效率有了较大的提升。

5 总结

本文提出了一种基于模型进化的测试数据生成方法，可以通过先建立初始模型，再按照一定的目标函数指导进化模型的生成，通过不断从新模型中生成测试用例，逐步提高待测应用测试覆盖率，最后，通过本方法构建了原型工具，自动化测试实例应用 Capture，也设计了对比实验与七种同类原型工具在 68 个开源 App 进行了测试，取得了良好的实验效果，相信通过对本模型的不断完善，诸如提高模型测试稳定性，以及兼容更多地安卓 SDK，生成的测试序列再组合[28]等，能在未来移动应用自动化测试中更多地被应用。

参考文献

[1] http://www.cnnic.cn

[2] Shauvik Roy Choudhary, Alessandra Gorla, and Alessandro Orso. 2015. Automated Test Input Generation for Android: Are We There Yet? (E). In 30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015. 429–440. DOI: http://dx.doi.org/10.1109/ASE.2015.89

[3] The Monkey UI android testing tool. http://developer.android.com/tools/help/monkey.html.

[4] A. Machiry, R. Tahiliani, and M. Naik. Dynodroid: An Input Generation System for Android Apps. In Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2013, pages 224–234, New York,NY, USA, 2013. ACM.

[5] S. Anand, M. Naik, M. J. Harrold, and H. Yang.

Automated Concolic Testing of Smartphone Apps. In Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, FSE '12, pages 59:1–59:11, New York, NY, USA, 2012. ACM.

[6] R. Mahmood, N. Mirzaei, and S. Malek. EvoDroid: Segmented Evolutionary Testing of Android Apps. In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2014, New York, NY, USA, 2014. ACM.

[7] D. Amalfitano, A. R. Fasolino, P. Tramontana, S. De Carmine,and A. M. Memon. Using GUI Ripping for Automated Testing of Android Applications. In Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, ASE 2012,

[8] T. Azim and I. Neamtiu. Targeted and Depth-first Exploration for Systematic Testing of Android Apps. In Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications, OOPSLA '13, pages 641–660, New York, NY, USA, 2013.ACM.

[9] S. Hao, B. Liu, S. Nath, W. G. Halfond, and R. Govindan. PUMA: Programmable UI-automation for large-scale dynamic analysis of mobile apps. In Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys '14, pages 204–217, New York, NY, USA, 2014. ACM.

[10] 徐宜生著.Android 群英传[M]..电子工业出版社 2015.

[11]Reto Meier 著,王鹏杰等译.Android 高级编程[M].清华大学出版社.2010.

[12] 谢晓园,徐宝文,史亮,聂长海.面向路径覆盖的演化测试用例生成技术(英文)[J].软件学报,2009,20(12):3117-3136.

[13] Dalal S R, Jain A, Karunanithi N,et al. Model-based testing in practice[C]. Proceedings of the 21st international conference on software engineering ACM，1999:285-294.

[14] Takala T, Katara M, Harty J. Experiences of System-Level Model-Based GUI Testing of an Android Application[C]// IEEE International Conference on Software Testing. IEEE Computer Society, 2011:377-386.

[15] Yang W, Prasad M R, Xie T. A Grey-Box Approach for Automated GUI-Model Generation of Mobile Applications[M]// Fundamental Approaches to Software Engineering. Springer Berlin Heidelberg, 2013:250-265.

[16] Choi W, Necula G, Sen K. Guided GUI testing of android apps with minimal restart and approximate learning[J]. Acm Sigplan Notices, 2013, 48(10):623-640.

[17] Google. 2017. Android UI Automator. (2017). Retrieved 2017-2-18 from http://developer.android.com/tools/help/uiautomator/index.html

[18] Young Min Baek and Doo-Hwan Bae. 2016. Automated model-based Android
GUI testing using multi-level GUI comparison criteria. In Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016, Singapore, September 3-7, 2016. 238–249.

[19] Deka B, Huang Z, Franzen C, et al. Rico: A Mobile App Dataset for Building Data-Driven Design Applications[C]// The, ACM Symposium. ACM, 2017:845-854.

[20] Memon A M, Soffa M L, Pollack M E. Coverage criteria for GUI testing[J]. Acm Sigsoft Software Engineering Notes, 2001, 26(5):256-267.

[21] Hong Zhu, Patrick A. V. Hall, and John H. R. May. 1997. Software Unit Test Coverage and Adequacy. ACM Comput. Surv. 29, 4 (Dec. 1997), 366–427.

[22] Liu J S. The Collapsed Gibbs Sampler in Bayesian Computations with Applications to a Gene Regulation Problem[J]. Publications of the American Statistical Association, 1994, 89(427):958-966.

[23] Eric Schkufza, Rahul Sharma, and Alex Aiken. 2013. Stochastic superoptimization. In Architectural Support for Programming Languages and Operating Systems, ASPLOS '13, Houston, TX, USA - March 16 - 20, 2013. 305–316.

[24] 王权. "马尔可夫链蒙特卡洛"(MCMC)方法在估计 IRT 模型参数中的应用[J]. 考试研究，2006(4):47-65.

[25] Soot Developers. 2017. Soot. (2017). Retrieved 2017-2-18 from https://github.com/Sable/soot

[26] Vlad Roubtsov. 2017. EMMA. (2017). Retrieved 2017-2-18 from http://emma.sourceforge.net/

[27] Saswat Anand. 2017. ELLA. (2017). Retrieved 2017-2-18 from https://github.com/saswatanand/ella

[28] 王进华,黄松,惠战伟,吴开舜.一种基于 GUI 模型的测试脚本组合方法[J].东南大学学报(自然科学版),2017,47(S1):164-169.

杨森 陆军工程大学软件工程硕士研究生，研究方向移动应用测试，并希望能将智能算法应用到移动应用的自动化测试中，对于软件可靠性分析，深度学习，云计算，建模方法等方向有极大的研究兴趣。。
联系手机:17626038075 邮箱：yangsen0310@qq.com