

Coomodroid: 通过使用案例组合为Android应用程序生成高质量的测试输入

Jue
Wang国家重点实验室专业
技术与计算机科学与技术
系, 南京, 中国南京, Zhi
mwang591@gmail.com

南京大学新型软件技术和计算机科
学与技术系科技科技技术与计算机科
学与技术系重点实验室, 中国南京Jy
y@nju.edu.cn

南京大学新型软件技术和计算机科
学与技术系科技科技能源科技科技
厅, 中国南京南京昌旭至九武.EDU.
CN

南京大学新型软件技术和计算机科
学与技术系科技科技技术和科技科技
科技州南京, 中国南京市南京市CA
OCHUN@NJU.EDU.CN

南京大学新型软件技术与计算机科
学与技术系南约硕士统一实验室,
南京XXM@nju.edu.cn

南京大学新型软件技术和计算机科
学与技术系剑路国家重点实验室,
南京, 中国LJ@nju.edu.cn

ABSTRACT

Android应用程序需要高质量的测试输入, 其一代仍然是开放的挑战。现有技术在探索复杂的应用程序功能中缺少只有长期, 有意义和有效的测试输入。观察这种测试输入通常可以分解成相对独立的短用例, 本文介绍了Combodroid, 一个根本不同的Android应用测试框架。CoffoOdroid获得用于表现出特定应用功能的用例 (手动提供或自动提取), 并系统地枚举使用情况的组合, 产生高质量的测试输入。Coffoyroid对现实世界应用程序的评估结果令人鼓舞。我们的全自动变量通过更多的代码 (APE仅优于2.1%), 占用了4.6%的最佳现有技术APE优势优势, 并且在广泛的测试对象中揭示了四个先前未知的错误。我们的半自动变体提高了使用少量手工劳动力的手动用例, 实现了白色盒子人体测试专家的可比覆盖率 (仅3.2%)。

KEYWORDS

软件测试, 移动应用程序

ACM参考格式: Jue Wang, Yanyan Jiang, Chang Xu, Chun Cao, Xiaoxing Ma和Jian Lu. 2020. CoffoOdroid: 通过使用案例组合为Android应用程序生成高质量的测试输入。在42届软件工程国际会议 (ICSE'20), 5月23日至29日, 2020年, 韩国首尔。ACM, 纽约, 纽约, 美国, 12页。https://doi.org/10.1145/3377811.3380382

如果没有收取副本或分发盈利或商业优势, 则授予个人或课堂上的所有或部分工作的所有或课堂工作的数字或部分工作中的所有或课堂使用的副本的许可 (课堂上) 授予, 并且副本承担本通知和第一页的完整引文。必须尊重由其他人拥有的本工作组件的版权必须尊重ACM。允许使用信用案。要复制另外或重新发布到服务器或重新分配给列表, 需要事先具体许可和/或费用。请求Permissions@acm.org的权限。ICSE'20, 5月23日至29日, 2020年, 首尔, 大韩民国 2020计算机协会。ACM ISBN 978-1-2503-7121-6 / 20/05 ... \$ 15.00
HTTPS://DOI.ORG/10.1145/3377811.3380382

1 INTRODUCTION

Android apps are oftentimes inadequately tested due to the lack of *high-quality test inputs*¹ to thoroughly exercise an app's functionalities and manifest potential bugs [11]. Existing automatic testing techniques fall short on exploring complex app functionalities that are only reachable by long and “meaningful” event sequences [33, 59]. Random or heuristic test input generation techniques [5, 6, 10, 28, 41–43, 56] can quickly cover superficial app functionalities, but have difficulty in reaching deeper app states to cover complex ones. Systematic input space exploration techniques [7, 47, 48, 61, 65] have severe scalability issues. Manual testing is effective and thorough, but also tedious, labor-intensive, and time-consuming, and usually hinders the rapid release of an app.

为了产生高质量的测试输入来彻底探索应用程序的功能, 我们观察到长而有意义的测试输入通常可以分解成相对独立的用例。用例是表现出指定应用程序的功能的短事件序列, 例如, 1切换设置, 2切换到活动, 或3下载Web内容。1→2→3是一个长 (且有意义) 的测试输入, 并且当应用程序在3中的行为在3个不同的设置时, 在显示不同的设置时尤其有用。

相反, 我们可以通过基本不同的两相方法来解决生长而有意义的测试输入的问题, 我们称之为Combodroid框架:

(1) 收集高质量的用例, 涵盖多种基本应用程序

(2) 连接许多用例以形成测试输入

可以使用案例手动提供 (例如, 由应用程序的开发人员) 或从执行跟踪中自动提取。由于开发人员清楚地了解如何实施要求, 因此它们可以轻松提供具有很少的手工劳动力的高质量用例。从执行迹线自动提取用例,

1在测试Android应用程序的上下文中, 测试输入是Android系统的原子输入事件 (触摸, 滑动等) 的序列。

我们利用使用案例的洞察，通过他们的定义，几乎在静止应用状态下开始和结束，通常是稳定的GUI。因此，我们设计了一种算法，可以自动识别这种GUI状态并从长截图序列中提取使用情况。

为了有效地生成高质量的用例组合（或简短的组合）作为测试输入，我们将算法设计为分类组合以获得最大化的测试分集。特别地，我们定义了与关系的对齐，这决定了在相同的静态状态下连接的两个用例，以preune可能无效的组合。我们还定义了依赖关系，它确定了用例是否会影响另一个的行为。我们仅生成对齐的组合，具有足够的数据流集分以获得有效的测试输入生成。

我们将这些想法作为Coffroid工具实施，包括 α 和半自动全自动组织机器人在测试场景中 fodroid 都是有效的：

- (1) 与最有效的现有技术APE [28]和最广泛使用的技术 α 猴[26]相比，平均涵盖了4.6%和6.7%。CoffoOdroid还揭示了广泛测试的主题中的四个先前未知的错误[51-54]。

促进了手动提供的用例的覆盖范围13.2%，与人类测试专家相比，实现了竞争规范覆盖率（间隙仅为3.2%），但劳动力较少。本文的其余部分安排如下。第2节概述了我们的方法，具有说明性示例。我们的方法的详细信息在第3节中讨论了第4节介绍了CoffoOdroid实施，我们的广泛评估是在第5节中进行的。第6节调查相关工作，第7条结束了本文。

2 OVERVIEW

图1显示了Combodroid工作流程。

CoffoOdroid拍摄了测试P下的应用程序，并重复由获取用例（左侧框）和枚举使用情况组合（右侧框）组成的两相位测试程序。我们使用激励示例来解释CoffoOdroid的工作流程，在AARD2中由Combodroid α 找到的先前未知的错误2（一个受欢迎的字典应用程序）。此错误需要长（且有意义）测试输入1→3→2→4→5触发。获取用例。我们首先观察到一个有意义的用例（事件序列）通常在静态应用程序状态下结束，其中应用程序是空闲的（在稳定的GUI上闲置（完成所有接收的事件）。静止状态自然表明人类可以在计算机人类交互中执行一个动作的下一步。在AARD2中，有用的用例包括添加/删除字典，搜索单词，查看单词的细节说明等。使用情况可以由人类显影剂（注意到CD B）提供。Coffodroid包含辅助工具，以帮助开发人员通过录制事件序列（UI和系统）来收集使用案例

2aard2已在现有研究中广泛评估[43,56,57]。然而，Combodroid α 是第一个揭开这个错误的。

事件[46]）在指定的时间间隔处。CoffoOdroid自动识别静态状态，并与使用情况一起收集执行迹线和GUI快照。在AARD2中，应用程序的开发人员将在提供有意义的用例时毫无困难，如1,2。。。, 5, (CID: 2)。

也可以通过自动分析应用现有的执行迹线（指出的CD）来提取使用情况。Combodroid在运行时在使用现有算法[10]的GUI转换时在运行时进行扩展标记的转换系统（elts）[31]。类似的稳定GUI被聚集成允许的单个状态。在执行跟踪中的一对稳定GUI之间的每个输入事件被添加为elts中的转换（用该事件标记）。

（因为首先没有跟踪），我们实现了基准DFS — 相似的状态空间探索工具[5]以生成初始测试迹线。允许唯一的独特的无环过渡路径被提取为可能的用例。在AARD2中，自动生成的用例不像手动那样可读，但共享类似的特征（例如，从静止应用程序状态开始和结束）。尽管如此，Combodroid成功地将不同的页面（例如，字典，搜索和详细页面）成功地确定为elts中的不同状态，而生成的用例涵盖了1,2中的所有功能。。。, 5, (CID: 2)。枚举用例组合。无论哪种方式，CoffoOdroid枚举了用例的组合（组合），以获得高质量的测试输入。组合是一系列用例

$$(u_1) \rightarrow (u_2) \rightarrow \dots \rightarrow (u_n)$$

where u_1 starts from the app's initial state. To make combos effective in testing, a combo should additionally satisfy:

(1) 可交付能力：对于所有 $1 \leq i < n$ ，UI与UI + 1对齐。对于U与V对齐，U中的最后一个GUI布局应该类似于v中的第一个（例如，它是在u之后立即向应用程序传递v的SANE）。相似性的特征在于基于编辑距离的测量。

(2) 数据流分集：UI至少取决于UI和 $i > j$ ，存在至少k个不同的对（UI，UJ）。为了依赖于v，应该有一些用于U中使用的共享程序状态并在v中进行修改。因此，我们过滤了松散连接的用例组合。

Combodroid中的系统枚举首先搜索数据相关对进行最大化的数据流分集，然后添加随机过渡用例以满足可传递性。在AARD2，1→2和4→5中是数据依赖性3。然后，CoffoOdroid将1→2→4→5作为骨架，其填充过渡用例（3和（CID: 2）），以产生图1中的错误触发组合（n = 8的组合，k = 2）。反馈循环。生成的Combos将使用正在收集的执行迹线传送到应用程序。交付后，如果没有在使用案例生成期间识别的那些以外的新探索的静态应用状态，CoffoOdroid终止。否则，CoffoOdroid重新启动第一阶段，要求人类有关这些状态的其他有效用例（例如，在执行期间访问它们），或者提取更有利可图

3使用案例删除字典（2）覆盖添加字典（1）中引用的字典对象，因此2取决于1。出于共享WebView对象的类似原因，5依赖于4。

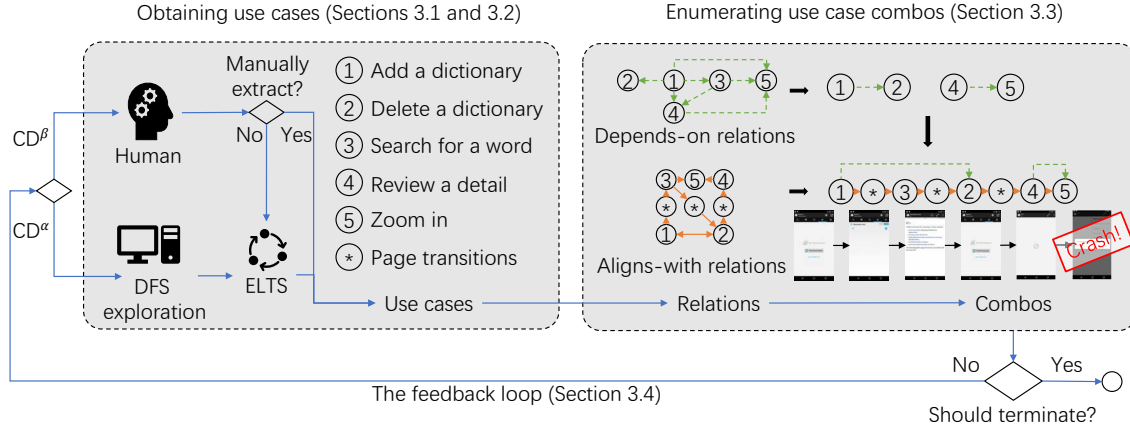


图1: Coildroid概述和激励错误示例

使用新收集的执行痕迹精制的允许壳。

虫子的表现。图1中的组合崩溃了该应用程序。删除字典后，将删除其所有细节字解释。但是，先前搜索的单词的“细节”页面仍然缓存在应用程序中。返回此类详细信息页面显示空白（空白）WebView。随后的缩放触发了nullpointerexcepti on的崩溃。所有八种用例（12个事件）都是必要触发错误的，并且现有技术不太可能产生这种长的事件序列，这确实在我们的评估中未能这样做。

3 APPROACH

3.1 Notations and Definitions

Given an Android app P , our goal is to generate high-quality test inputs via use case combinations. Android apps are GUI-centered and event-driven. The runtime GUI layout (snapshot) ℓ is a tree in which each node $w \in \ell$ is a GUI widget (e.g., a button or a text field object). We use $w.type$ to refer to w 's widget type (e.g., a button or a text field). When P is inactive (closed or paused to background), there is no GUI layout and $\ell = \perp$.

事件E
= (CID: 6) T, R, Z (CID: 7) 是一个记录，其中E.T, E.R和E.Z分别表示E的事件类型，接收器窗口小部件和相关数据。一个事件可以是UI事件或系统事件，而T的示例是“ui-click”，“ui-swipe”或“sys-pause”。对于UI事件，接收器R ((CID: 2)) = W表示E在运行时可以传递给W ∈ (CID: 2)。R ((CID: 2)) = \perp 表示无法传递此事件。系统事件的接收器始终是“系统”窗口小部件。其他事件特定信息存储在z，例如，在文本字段中输入的文本或添加文件的内容。

使用一系列事件E =

[E1, E2, ..., en]产生执行跟踪 τ =执行utep (e) = (cid : 6) l, m, t (cid: 7)。如算法1, l4, m和t所定义的，表示diped
gui布局，方法调用跟踪和每个事件的相应方法调用。

4对于l = [1, χ 2. ..., n+ 1], i是在e中的第一个I - 1事件之后的GUI布局转储（在静态状态）发送到应用程序。

算法1: 执行一系列事件

```

1 Function Executep(E)
2    $\ell \leftarrow \text{GetGUI}(); L \leftarrow [\ell]; M \leftarrow \emptyset; T \leftarrow \emptyset;$ 
3   for each  $e \in E$  do
4     if  $r(\ell) \neq \perp$  then // e can be sent on  $\ell$ 
5        $M' \leftarrow \text{SendEventToApp}(e.t, e.r(\ell), e.z);$  // send
        event e to P, wait for a quiescent state, and return the
        corresponding method invocation sequence
7        $M \leftarrow M \cup M'; T \leftarrow T \cup \{(e, M')\};$ 
8        $\ell \leftarrow \text{GetGUI}(); L \leftarrow L \cup [\ell];$ 
9     else
10      return  $\perp$ ;
11  return  $\langle L, M, T \rangle$ ;
```

$U = [E1, E2, \dots, e \mid U$

]]也是一个事件序列。对于人类开发人员来说，它是简单的，以便以任一向手动提供用例：（1）注释用例作为执行跟踪 τ ，或（2）馈送 τ 的子字符串到以下自动提取算法。

3.2 Use Case Extraction

在挖掘的扩展标记的过渡系统[31] (elts) 上提取用例。此外，在没有提供跟踪的全自动设置中，我们使用标准深度探索来获取自动启动跟踪。

挖掘自动机。给定执行跟踪 $\tau = L, m, t$ 从执行事件序列e，其对应的elts是一个threTuple g
= $\langle s, e, \Delta \rangle$ ，其中s是一组抽象状态 ($\{s \mid s \in S\}$ 是GUI布局L) 的分区， $\Delta: S \times e \rightarrow S$ 包含状态转换。我们采用SWIFTHAND

[10]中的现有算法用于挖掘一个最小的允许将类似的GUI布局组在一起，即等效 (1, 2) 5对所有GUI布局 1, 2相同

5.使用AMOLA [6]的LV.4

GUI比较标准 (GUICC) 测量GUI之间的相似性，即GUI布局 1和 2是等同的，当虽然 $\in \text{eer} (1)$ $\text{er} (2)$ 。

Algorithm 2: ELTS Mining

```

1 Function
  MineELTS( $\langle L = [\ell_1, \ell_2, \dots, \ell_{n+1}], M, T \rangle, E = [e_1, e_2, \dots, e_n]$ )
2    $S \leftarrow \{\{\ell\} \mid \ell \in L\}$ ; // initially, no state is merged
3    $\delta \leftarrow \ell, e, \ell_{+1} \quad 1 \leq i \leq n$ ;
4 对于每个  $(S_i, S_j) \in S \times S$  和  $S_i \neq S_j$  DO //在BlueFring中

5      $\langle S', \delta' \rangle \leftarrow \text{merge-recursive}(s_i, s_j, S, \delta)$ ;
6     if  $\langle S', \delta' \rangle \neq \perp$  then
7        $\langle S, \delta \rangle \leftarrow \langle S', \delta' \rangle$ ; // update merged states
8   return  $\langle S, E, \delta \rangle$ ;

9 Function merge-recursive( $s, t, S, \delta$ )
10  if  $\forall \ell_1 \in s, \ell_2 \in t. \text{equivalent}(\ell_1, \ell_2)$  then
11     $S' \leftarrow S \setminus \{s, t\} \cup \{s \cup t\}$ ;  $\delta' \leftarrow \delta[s/t]$ ;
12    for each  $\langle s, e, t_1 \rangle, \langle s, e, t_2 \rangle \in \delta$  where  $t_1 \neq t_2$  do
13       $\langle S', \delta' \rangle \leftarrow \text{merge-recursive}(t_1, t_2, S', \delta')$ ;
14      if  $\langle S', \delta' \rangle = \perp$  then
15        break;
16    return  $\langle S', \delta' \rangle$ ;
17 return  $\perp$ ; // merging failed

```

状态。这种算法（算法2）最初用于Android应用的动态模型提取。提取用例。有效的路径= $[S_0, S_1, \dots, S_M]$ $\text{ONG}(S, E, \Delta)$ 其中 $\delta(S_{i-1}, E_i) = S_i$ 为所有 $1 \leq i \leq M$ 自然对应于事件序列

$$u = [e_1, e_2, \dots, e_m]$$

as a likely use case. Therefore, the automatic use case extraction algorithm enumerates all acyclic paths in G and produces a use case for each of them.

请注意，我们的自动算法可能从elts中提取可能使用案例。以这种方式，我们可以最大限度地提高所有可能的用例的机会。此外，最有可能的情况可能是真正的用例，而其他可能会与它们共享类似的功能（例如，从静止应用程序状态开始和结束），也可以有效探索应用程序的行为。引导使用案例生成。在Combodroid的全自动设置中，使用与A3E算法类似的标准DFS - 相似的状态空间探索策略引导用例提取[5]。

从初始状态开始，我们采取GUI布局快照（CID: 2），分析所有小部件 $W \in (\text{CID: 2})$ ，以获取 W 的所有可能操作。对于每个动作（例如，单击一个按钮，或从预定义字典中输入随机文本到文本字段[43]），我们创建一个事件 E_6 并将其添加到 EUI 。然后，我们顺序地执行（将事件发送到应用程序并等待静态状态） $EUI \cup \text{esys}$ 中的所有事件，其中 esOs 是一组预定义的系统事件。如果执行事件达到未开发的GUI（CID: 2）（CID: 9）执行回溯（因此，勘探递归（CID: 2）（CID: 9）生了一系列事件 e

6对于 $E = T, r, z$ ， $e.t$ 和 $e.z$ 是简单的以确定。接收器 $E.R$ 由稍后在第3.3节中描述的基于编辑距离的算法确定。

3.3枚举用例组合

假设用例 $U =$

$\{U_1, U_2, \dots\}$ ，通过执行事件序列 E 从执行跟踪 $\tau = L, m, t$ 中提取 $UN\}$ 。使用案例组合（或组合）是由 $[UI_1 \rightarrow UI_2 \rightarrow \dots \rightarrow UI_k]$ 。顺序地连接组合的使用情况下的事件产生可运行的测试输入。不幸的是，随机生成的组合通常在执行中早期停止，因为可能存在在输送时间GUI上没有接收器的事件 E ，即， $E.R(\quad) = \perp$ 。考虑激励例中的组合 $2 \rightarrow 5$ （图1）。删除字典后，“放大”事件在“变焦”事件中没有接收器，因为当前GUI不包含包含Zoomin按钮的ListView菜单。为了产生高质量的用例组合，我们利用以下两种用例关系：

对齐。对于两个用例 $U = [E_1, E_2, \dots, e_n]$ 和 $v = [e'_1, e'_2, \dots, e'_m]$ ，我们说 U 与 v 或 $u \sim v$ ，如果我们在 en e'_1 可以成功交付一旦我们见证了。换句话说， $U \neq v$ 如果 $e'_1.r(\quad n) = \perp$ 在痕迹 τ 中执行 $en \in e$ 之后的GUI布局。在用例对齐（并重放事件序列）的另一个问题是确定如何将UI事件 E 传送到特定的GUI布局。对于 $\tau = \{W_1, W_2, \dots, W_n\}$ 在 τ 中发送之前的GUI布局，以及任意 $W'_i = \{W'_1, W'_2, \dots, W'_i\}$ ，我们知道存在 $1 \leq i \leq n$ 这样 $e.r(\quad) = w_i \in \tau$ ，因为 w_i 是 τ 的接收器小部件。因此，窗口小部件 $w'_j \in e'$

这是“最相似”的 w_i 应该是 e 上的 e 的接收器，即， $e.r(\quad) = w'_j$ 。为了测量GUI布局之间的相似性，我们使用Repdroid中的算法计算 τ 和 τ' 之间的编辑距离[68]。我们找到最短的编辑操作序列（每个编辑操作都插入或删除窗口小部件），其将 τ 变换为 τ' 。如果在转换期间未删除 w_i ，则它必须具有唯一的对应关系 $w'_j \in \tau'$ 。因此，我们让 $e.r(\quad) = w'_j$ ；否则，否则会被删除， $E.R(\quad) = \perp$ 。

取决于。对于用例 U 和 V ，我们说 V 取决于 U ，或 $U \neq v$ ，如果两个用例潜在地依赖于数据。数据依赖性以方法级别测量。考虑到 τ 中的方法调用迹线，如果为 $e'u$ 的 $e'u$ 和 $m' \in t(e')$ 存在一个方法 $M \in T(e)$ ，例如 m' 数据取决于 m ，则说明 $u \in V$ 。方法之间的数据依赖性由轻量级静态分析确定。 m' 数据 - 如果存在（摘要）对象或在 m' 中读取的读访问权限，则依赖于 m 。

组合生成。与之对齐和依赖关系指导我们使用案例组合（组合）生成。为了最大化生成的组合的多样性，我们强制执行每个组合 $c = [u_1 \rightarrow u_2 \rightarrow \dots \rightarrow U \mid c]$ 满足：

- (1) 每个组合是一个独立的测试用例： $e_1.r(\chi^0) = \perp$ $0 \leq i$ 是应用程序的初始GUI布局和 E_1 是 U_1 中的第一个事件；(2) 组合中的连续用例对齐： $UI_i \sim UI_{i+1}$ 对于全部 $1 \leq i < |C|$ ；和

Algorithm 3: Combo Generation

```

1 Function RandomCombo( $U, \ell_0, k$ )
2    $G \leftarrow V, E$  randomDAG( $2k$ ); // random DAG of  $E = 2k$ 
3    $\mathcal{F} \leftarrow \{ (v, \text{randomchoice}(u)) \mid v \in V \}$ ; // 随机分配每个  $v \in V$ 
   用例
4
5   for each linear extension  $[v_1, v_2, \dots, v_{|V|}]$  of  $G$  do
6     for  $u_0 = [e_1, e_2, \dots, e_m] \in U \wedge e_1.r(\ell_0) \neq \perp$  do
7        $c \leftarrow \text{connect}(u_0, F(v_1), U, 0) ::$ 
        $\text{connect}(F(v_1), F(v_2), U, 0) :: \dots ::$ 
        $\text{connect}(F(v_{n-1}), F(v_{|V|}), U, 0) :: [F(v_{|V|})];$ 
       // add paddings such that consecutive use cases are
       aligned
8       if  $\perp \notin c$  then
9         return  $c$ ;
10  return  $\perp$ ;
11 Function connect( $u, dst, U, depth$ )
12  if  $u \rightsquigarrow dst$  then
13    return  $[u]$ ;
14  if  $depth > \text{MAX\_DEPTH}$  then
15    return  $\perp$ ;
16  for  $u' \in U \wedge u \rightsquigarrow u'$  do
17     $seq \leftarrow \text{connect}(u', dst, U, depth + 1)$ ;
18    if  $seq \neq \perp$  then
19      return  $[u] :: seq$ ;
20  return  $\perp$ ;

```

(3) 在组合中的用例展示 k -data-flow 分集, 即存在 k 个不同的 (U_i, U_j) ($1 \leq i < j \leq |c|$), 使得 $U_i \not\sim U_j$ 。

用于生成组合的算法以算法3呈现。给定一组用例 U , 应用程序的初始GUI布局 (CID: 2) 0, 以及数据流分集度量 K , 首先采样随机骨架。骨架是一个定向的非循环图 $G(v, e)$, 其中 $|e| = 2k$ 。如果 G 的数据流分集小于 K (第4行), 则应重新启动生成。否则, 每个顶点 $v \in V$ 被分配在 U 中的随机用例 $F(v)$ (第2-3行);

骨架 G 的衬垫扩展对应于一系列用例: $[F(v_1), f(v_2), \dots, f(v_{|V|})]$ 。我们尝试用更多用例用蜂窝使用用例 $f(v_i)$ 和 $f(v_i + 1)$ ($1 \leq i < |V|$) 以获得组合 c , 使 C 的连续用例是对齐 (第7行)。填充用例是深度首先搜索的最大长度限制 MAX_DEPTH (第11-20行)。

我们还在 C (第6行) 中的第一个用例之前添加拼接, 使得得到的组合可以传递给初始应用状态 (因此 C 可以用作独立的测试用例)。如果所有上述拼接存在, 我们成功获得了满足我们要求的用例组合 (第8-9行)。这样的组合被发送到应用程序进行测试。

7A在每对GUI之间的过渡自然存在于精心设计的应用程序中。因此, 它非常类似于所有上述镶边。

3.4 Feedback Loop of ComboDroid

如图1所示, 可以有多个使用案例生成和组合枚举的迭代。当枚举组合发送到应用程序并发现以前未知的状态时, 应启动新的迭代。在下次迭代开始之前, 开发人员可以手动检查测试报告并提供/注释更多用例。假设我们在使用案例生成和组合枚举的所有先前迭代中连接执行跟踪。在概念上, 这可以被视为在发送组合中的所有事件之后添加额外的“重新启动”事件。这种合并的痕迹用于允许采矿和在下一轮迭代中的用例提取。

4 IMPLEMENTATION

The ComboDroid framework is implemented using Kotlin and Java. ComboDroid consists of a fully automatic variant ComboDroid ^{α} and a semi-automatic variant ComboDroid ^{β} . We extensively used open-source tools in the implementation, and ComboDroid is also open-source available⁹: GUI events are recorded by Getevent [25]; GUI and system events are delivered using Android Debug Bridge (ADB) [22]; GUI layouts are dumped by Android UI Automator [24]; method traces are collected by program instrumentation with Soot [13]. The implementation follows the descriptions in Section 3. We follow the common practice of existing state-of-the-art techniques [6, 28, 43, 56] and identify quiescent app states by stabilized GUIs. Specifically, we take the same implementation as APE [28] by dumping GUI layouts every 200ms until it is stable (using the Lv.4 GUICC) with a 1000ms upper-bound.

, 我们实现了DFS-Alike

Exploration工具, 并通过重播以前的执行跟踪来遵循与APE [28]相同的实现。对于ComboDroid, 我们分析了人类测试仪将每个事件发送的GUI布局以及相应的事件来确定每个事件的接收器。

在轻量级静态分析确定依赖关系中, 我们还模拟了Android 6.0

API (API级别23) [23]以确定对资源的读/写访问, 例如, 我们确定SQLiteDatabase.execSQL中的SQL命令是否为读或写入数据库。此外, 对于(摘要)对象, 如果名称与正则表达式匹配的任何方法

```
(get|is|read)(.+)|on(.+)changed
```

被称为, 我们认为这是一个读; 同样, 调用任何名称匹配的方法

```
(set|write|change|modify)(.+)
```

被认为是写作。在每次迭代中设定了对组合 α 的深度首先探索。在组合生成 (Combodroid α 和 Combodroid β) 中, 我们设置数据流分集 $k = 2$ 和 $\text{max_depth} = 5$ 。如果存在 d 依赖于边缘, 我们会生成 $D2$ 随机组合 (通过随机 Comcombo)。

允许在允许挖掘后添加了8A重启事件。9https://github.com/skull591/combodroid-artifact。

5 EVALUATION

本节介绍了我们对Cobocodroid的评估。在5.1节中描述了实验对象和设置，然后分别在5.2和5.3节中评估了组合 α （全自动变体）和Combodroid β （人辅助变体）的评价结果。在第5.4节中提出了讨论包括有效性的威胁。

5.1 实验科目和设置

The first column of Table 1 lists the 17 evaluation subjects. The apps are selected using the following rules: First, we selected the three largest (in LoC) apps evaluated in existing work [28, 43, 56]: WORDPRESS, K-9 MAIL, and MYEXPENSE. Second, we randomly selected nine apps with at least 10K Downloads evaluated in the existing work [28, 43, 56]: WIKIPEDIA, ANKIDROID, AMAZEFILEMANAGER, ANYMEMO, HACKER NEWS READER, CALLMETER, AARD2, WORLD CLOCK, and ALOGCAT. Additionally, we randomly selected five popular (at least 100 stars by 2018) open-source apps from Github: ANTENNAPOD, POCKETHUB, SIMPLETASK, SIMPLE DRAW, and COOLCLOCK.

如果在没有适当的初始设置（例如用户登录）的情况下无法访问应用程序的主要功能，我们提供了应用程序脚本以完成设置。所有评估的技术都接收到完全相同的脚本（一旦达到初始设置GUI，脚本会自动运行）以确保公平比较。我们没有嘲笑初始设置脚本以外的任何进一步的功能。

我们使用两项指标来衡量测试彻底性。第一个是jacoco [32]收集的Bytecode指令覆盖，因为更高的代码覆盖率与更好的应用功能的功能强烈相关。其次，我们研究技术是否可以通过检查Android系统的日志来表现出先知或未知的错误。

，我们将其与Theal-theal-of-theal-of-of-theal-of-the-of-theal-of-termate: monkey [26], sapienz [43], 和ape [28] 10。对于每个主题，我们运行了每次自动测试技术12小时，以模拟夜间连续集成构建和最多12小时。循环。我们为每个主题进行了Combodroid，我们运行了每种技术三次并报告了平均结果。然后研究了测试覆盖和错误表现结果。

，我们将其与人类专家进行比较。，我们给出了Refor手动用例，为每个测试对象的一个工作日（8个工作时间）进行了反转手动用例，然后每次进行12小时。同时，我们询问另一位独立招聘的Android检测专家（一名研究生曾在Android Apps的测试和分析上发表过一些研究论文），以便在三个工作日的时间限制（24个工作时间）。人类专家可以访问应用程序的源代码，并被告知使用覆盖反馈来最大限度地提高代码覆盖率。由于手工劳动不可扩展，我们只评估了前10个LOC的主题，如表3所示。为了减少分心（确认和诊断

错误是耗时的），我们要求人类专家提供任何错误报告。因此，在对组织的评估中仅研究了测试覆盖。所有实验均在Octa-Core Intel I7-4790U PC上进行，其中16个GIB RAM运行Ubuntu 16.04 LTS和Android 6.0仿真器。

5.2 Evaluation Results: ComboDroid α

表1和表2中列出了12小时的覆盖结果和表现错误。详细的覆盖率趋势绘制在图2中。这些结果与最近的实证研究一致[59]：自动化技术，到那个时间几乎不佳，最简单的猴子。最好的现有技术，APE，通过更多的代码占用2.1%的猴子略微优于猴子。鼓励，Coffodroid α 在几乎所有主题11中始终如一地优于现有技术。对于Alogcat, Coolclock和Callmeter, CoffoOdroid α 在12小时内终止，而其他受试者在超出时限直至其超出。与最佳现有技术猿相比，Combodroid α 平均覆盖了4.6%的代码。这种改进甚至是2倍，就像猴子的改善一样多。考虑到APE实现产生1.5倍的事件在12小时内（对于APE的Combodroid α V.S. 300K），Combodroid α 在利用每个事件的优点方面有很大效率。图2中的渐进覆盖显示了CoffoOdroid α

通常开始在六个小时后表现出现有的技术。考虑到目前的Combodroid α 实现在1/2速度发出事件，结果也有望。此外，在最后几个小时内，现有技术的代码覆盖率通常是边缘（或零）。相比之下，Combodroid α 一直探索有用的用例组合来涵盖更多代码。错误表现评估结果（表2）也令人鼓舞。我们手动检查了所有技术的测试日志，并找到了具有显式根源的12个可重复错误。除了简单的绘制中的错误（由于实施限制，CoffoOdroid α 错过了它），Combodroid α 表现出所有11个以前未知的或未知的错误，其中最好的现有技术APE，表现出7（64%）。我们还报告了四个以前未知的错误（APE可以发现其中两个）到开发人员。所有这些都证实了，其中两个已经固定。此外，由Combodroid α 唯一发现的两个先前未知的错误是深度突发，需要长（和有意义）的输入序列来触发。激励例子（图1）是这样的案例。因此，我们持有强大的证据表明，与现有技术相比，CoffoOdroid α 更有效地为Android应用程序提供高质量的测试输入。

β

Cobodroid β 的评估结果显示在表3中。对于所有受试者，Coffroid β 率直至超过时限。预计人类测试专家明显优越

11ape和Combodroid α 与猴子相比，简单抽奖的代码较少，因为实现不识别Canvas小部件，因此不会发送拖动事件。我们认为这是一个实施限制。

¹⁰SINCE APE [28]显著优于替代品（其他相关工作），我们没有显示本文的其他技术的结果。

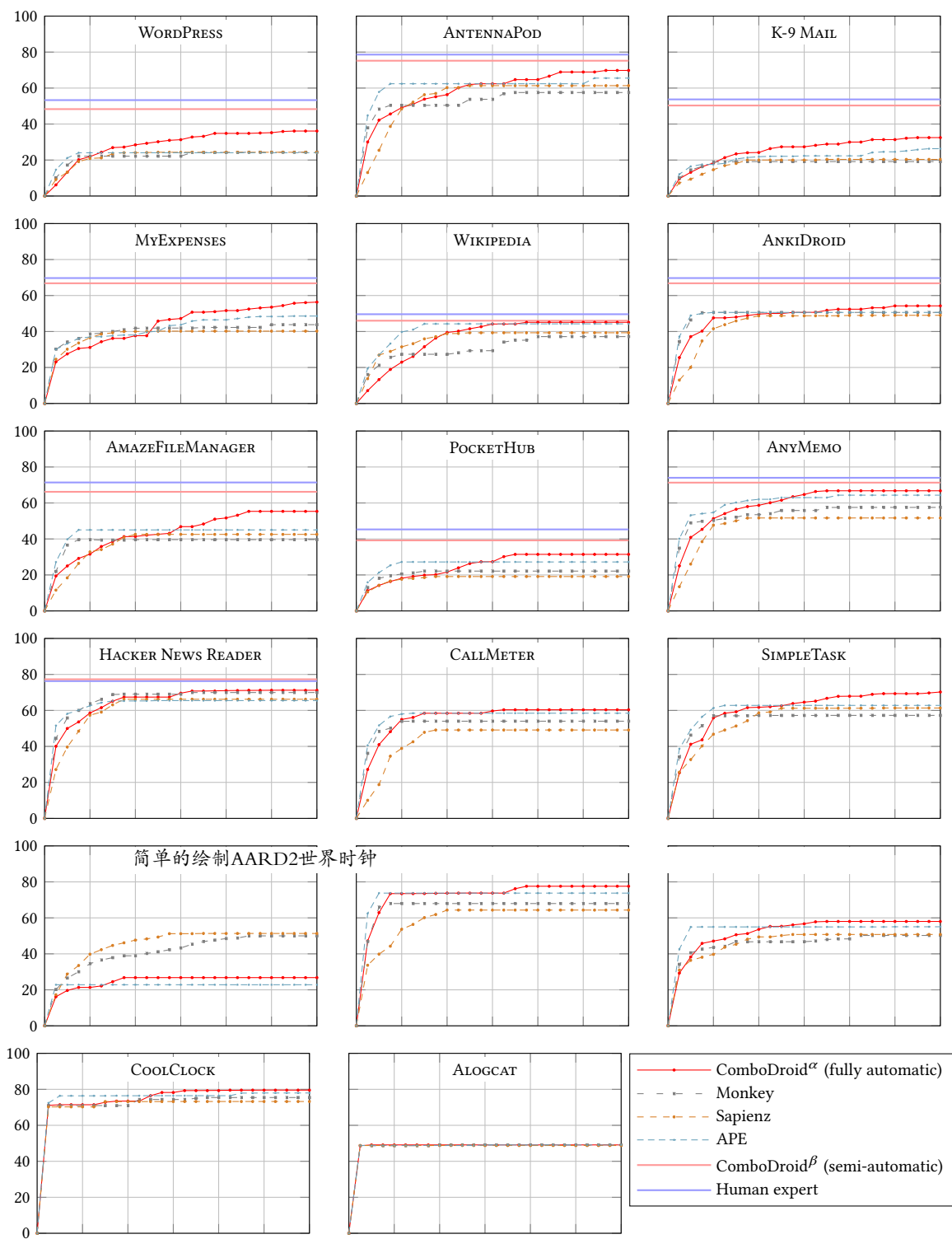
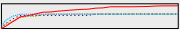
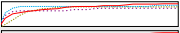


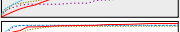

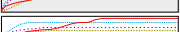
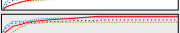


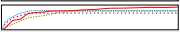

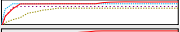

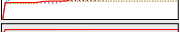




图2: 评估技术的渐进覆盖报告 (平均三次运行)。x轴是花费的时间 (0-12小时)。Y轴表示迄今为止所涵盖的代码百分比。

表1: Coffroid α 的评估结果：测试覆盖

主题（类别，下载; LOC）猴子Sapienz APECombodroid α 覆盖率						
WORDPRESS, WP (Social, 5M–10M; 327,845)	24.4%	24.3%	24.1%	36.1% (+11.7%)		
ANTENNAPOD, AP (Video, 100K–500K; 262,460)	57.5%	61.3%	65.5%	69.8% (+4.3%)		
K-9 MAIL, K9 (Communication, 5M–10M; 159,708)	19.1%	20.4%	26.3%	32.5% (+6.2%)		
MYEXPENSES, ME (Finance, 500K–1M; 104,306)	43.8%	40.2%	48.6%	56.3% (+7.7%)		
WIKIPEDIA, WIKI (Books, 10M–50M; 93,404)	37.2%	39.3%	44.3%	45.1% (+0.8%)		
ANKIDROID, AD (Education, 1M–5M; 66,513)	50.6%	49.0%	50.6%	54.3% (+3.7%)		
AMAZEFILEMANAGER, AFM (Tools, 100K–500K; 66,126)	39.6%	42.5%	45.0%	55.2% (+10.2%)		
POCKETHUB, PH (Tools, 100K–500K; 47,946)	22.1%	19.1%	27.2%	31.4% (+4.2%)		
ANYMEMO, AM (Education, 100K–500K; 40,503)	57.5%	51.7%	64.3%	66.8% (+2.5%)		
HACKER NEWS READER, HNR (News, 50K–100K; 38,315)	69.9%	66.2%	65.5%	71.2% (+1.3%)		
CALLMETER, CM (Tools, 1M–5M; 21,973)	54.0%	49.1%	58.5%	60.4% (+1.9%)		
SIMPLETASK, ST (Productivity, 10K–50K; 20,980)	57.2%	57.2%	62.8%	70.2% (+7.4%)		
SIMPLE DRAW, SD (Tools, 10K–50K; 18,685)	50.0%	51.3%	22.8%	26.8% (-24.5%)		
AARD2, AARD (Books, 10K–50K; 9,622)	68.0%	64.3%	73.8%	77.6% (+3.8%)		
WORLD CLOCK, WC (Bussiness, 1M–5M; 7,181)	50.2%	50.8%	55.1%	58.0% (+2.9%)		
COOLCLOCK, CC (Tools, 10K–50K; 2,762)	75.4%	73.2%	78.0%	79.6% (+1.6%)		
ALOGCAT, ALC (Tools, 100K–500K; 846)	49.1%	48.8%	49.1%	49.1% (0.0%)		
Average	48.6%	47.6%	50.7%	55.3% (+4.6%)		

—1列覆盖趋势绘制每个工具的覆盖率趋势。红色实线表示Combodroid α，虚线是现有的技术。详细的覆盖趋势如图2所示。括号中的数字是Coffroid α 与最佳现有技术（猴子，Sapienz和APE）之间的覆盖差异。

表2: CoffoOdroid α 的评估结果：Bug表现

发现错误ID原因		
WP-10147	Infinite recursion	APE, CD ^α
AP-3195	NULL指针取消引用所有K9-3308不匹配的MIME类型SAPIENZ	α
AFM-1351	NULL指针取消引用所有AFM-1402生命周期事件Mishandling	APE, CD α
AM-503*	Null pointer dereference	APE, CD ^α
CM-128*	Text input mishandling	APE, CD ^α
SD-49	Miss-used local variables	Monkey
AARD-90*	Null pointer dereference	CD ^α
AARD-7	Null pointer dereference	all

Monkey: 4 (33%); Sapienz: 4 (33%); APE: 7 (58%); CD^α: 11 (92%)

—1错误ID是项目GitHub存储库中的问题ID。一个出售的错误id 表示先前未知的错误。

自动化技术。即使是到目前为止最佳的自动化技术，Combodroid α 也覆盖了12.0%的代码。然而，当人类知识集成到我们的框架中，这种差距减少到3.2%：使用案例组合另外涵盖的代码仅比手动用例更多地覆盖13.2%。combodroid β。

大大放大了用例（覆盖47.5%的代码，甚至比Combodroid α 小4.4%），几乎与良好一样

表3: CoffoOdroid β 的评估结果：测试覆盖率

Subject	UC	CD ^α	ComboDroid ^β	Expert
WP (328K)	40.1%	36.1%	48.3% (+8.2%/+12.2%)	53.3% (+5.0%)
AP (262K)	65.4%	69.8%	75.2% (+9.8%/+5.4%)	78.6% (+3.4%)
K9 (160K)	38.5%	32.5%	50.3% (+11.8%/+17.8%)	53.7% (+3.4%)
ME (104K)	53.1%	56.3%	66.8% (+13.7%/+10.5%)	69.7% (+2.9%)
WIKI (93K)	37.3%	45.1%	46.0% (+8.7%/+0.9%)	49.6% (+3.6%)
AD (67K)	50.3%	54.3%	66.8% (+16.5%/+12.5%)	71.4% (+4.6%)
AFM (66K)	43.3%	55.2%	66.2% (+22.9%/+11%)	67.3% (+1.1%)
PH (48K)	31.5%	31.4%	39.2% (+7.7%/+7.8%)	45.3% (+6.1%)
AM (41K)	62.1%	66.8%	71.3% (+9.2%/+4.5%)	74.0% (+2.7%)
HNR (38K)	53.4%	71.2%	76.5% (+23.1%/+5.3%)	76.3% (-0.2%)

Average 47.5% 51.9% 60.7% (+13.2%/+8.8%) 63.9% (+3.2%)

—1列主题中的数字是应用程序的位置。专栏UC，CD α，Combodroid β 和专家分别显示手动用例，Combodroid α，Combodroid β 和人类专家的代码覆盖。柱子组合 β 括号中的数字分别表明了Coffroid β 和手动用例和组合 α 之间的覆盖差异。列专家括号中的数字表明人类专家和结合 β 之间的差异。

作为人类专家。令人惊讶的是，Coffroid β 甚至优于黑客新闻读者的人类专家。在分析代码和覆盖数据后，我们发现黑客新闻读卡器可以在设置中启用新闻文章的数据预载。启用后，以应用程序内部格式打开文章

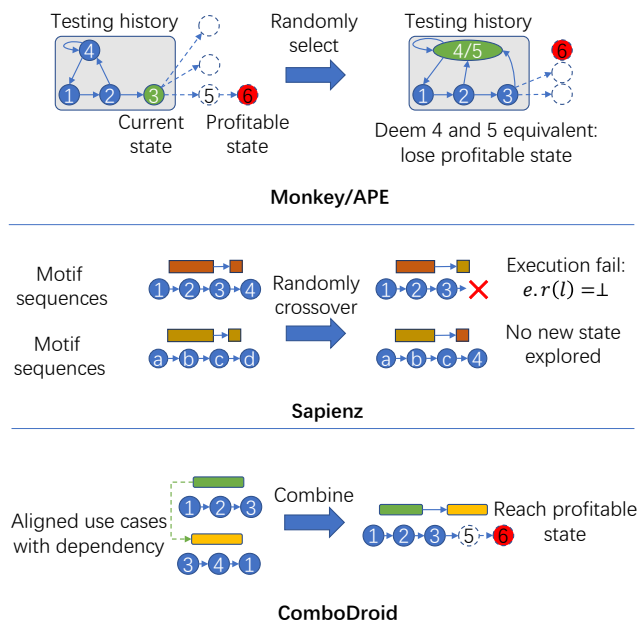


图3：评估技术中的状态空间探索策略的定性插图。

调用其他代码以处理预加载的数据。人类专家错过了这种微妙的依赖；另一方面，虽然也没有被任何单一手动用例涵盖的，但是combodroid β

正确地确定了这样的数据依赖关系（在依赖关系中），并因此产生了用例组合。虽然在初步阶段，Combodroid β 展示了在测试Android应用程序中自动利用人类洞察的潜力，以便在互补和促进自动化技术方面。

5.4 Discussions

5.4.1 Towards Thorough Automatic Testing of Android Apps. Figure 3 illustrates the search strategies of the evaluated techniques, for giving a qualitatively explanation of why ComboDroid $^{\alpha}$ outperformed existing techniques.

基于随机的技术猴子[26]和APE

[28]在每次向应用程序提供一个事件，因此完全没有意识到剩余状态空间。它们的局限性很明显：搜索策略纯粹基于嘈杂的探索历史。此类策略可能会在随机尝试时容易地失去深度（和有利可图）的应用状态（例如，按按钮返回应用程序的主菜单）。

SAPIENZ

[43]，但在引导搜索中利用主题序列，不能有效地组装它们。首先，没有基本序列的理由或质量保证——它们是或多或少的随机事件序列。其次，遗传搜索中的突变和交叉操作在创建有用的主题序列组合中的效率低下：随机连接两个事件序列主要导致无用的组合。在长期运行中，Sapienz甚至覆盖的代码比猴子更少的代码并不令人惊讶。该结果与现有研究一致[59]一致。

相比之下，CoffoDroid α 产生了高质量用例和它们的组合，因此在覆盖应用程序功能方面非常有效，即使它达到60%的事件，也是如此。

与手动测试相比，自动测试仍然不那么令人满意：人类专家平均覆盖了12.0%的代码，而不是Combodroid α 。Combodroid β 的评估结果表明，这种差距主要是由于使用质量案例。我们的用例提取算法根本无法“了解”应用程序的功能和语义，然而，即使对于应用程序用户，也是有意义的用例非常自然。通过大型应用程序使用数据集的机器学习可能是解决此问题的有希望的方向。

5.4.2 利用Android应用的半自动测试的人力见解。 CoffoDroid成功地“放大”手动用例，实现与人类专家相比的竞争覆盖范围：增加更多人类援助促进测试彻底性。这部分验证了我们的直觉，即人类擅长草图绘制应用程序的主要功能；一旦提取了这样的见解（用例），可以将繁琐和重复的工作卸载到机器上。

作为一种概念原型，开启了对Android应用程序的人机协作测试的新研究方向。自动生成有意义的（和方便）的建议（通过程序分析或机器学习）来帮助手动测试仪，开发人员，甚至用户提供更好的使用情况是一个有益的未来方向。

5.4.3 有效性的威胁。 所选科目中的偏见。所选测试对象的代表性会影响我们结论的保真度。为了减轻这种威胁，我们选择了来自各种来源的评估科目：在现有工作中评估的流行基准加上GitHub中的随机性。这些受试者（1）大小（平均约76kloc），（2）维护良好（平均含有数千个修订和数百个问题），（3）流行（所有有10k+下载），并（4）各种类别。由于组织始终如一，并且在所有这些基准中始终如一地优于现有技术（由于实施限制而简单的简单），结论是结论的结论

比现有技术更有效。

随机性和非确定性。评估的技术（包括Cobicodroid）涉及随机性，受试者可能是非确定性的。因此，对于每个主题和技术，我们报告了三个独立运行的平均结果在与相同的设置（实验成本超过2,400多个CPU小时）下进行缓解此问题。

人为因素。人体测试人员的表现变为人员。因此，Cobodroid β 的评估结果

仅适用于该人类测试专家。由于毕业后Android测试/分析专家提前了解我们，我们确信他/她尽可能多地覆盖尽可能多的代码。

6 RELATED WORK

已经提出了许多技术用于Android应用程序测试的输入生成，包括全自动和半自动的输入。此外，一些技术为GUI / Web测试生成测试输入也与Coffroid共享相似之处。

Android应用程序的全自动测试输入生成。大多数现有技术旨在完全自动为Android应用程序生成测试输入。其中许多生成用于一般测试目的的测试输入。

随机测试是一种轻量级和实用的方法，其中大量随机事件被快速馈送到应用程序，包括猴子[26]，dynodroid[41]，Droidfuzzer[64]，Intentfuzzer[63]等。

使用GUI模型（预定义或开采）可以指导应用程序的状态空间的探索。代表性工作包括Mobiguitar[3]，Swifthand[60]，Amola[6]和最先进的猿类[28]。这种状态空间探索通常是通过深度（宽度）-漏洞搜索，例如A3E[5]，GAT[61]和EHBDRDROID[55]进行。但是，即使使用模型，现有技术也会在生成（和有意义）的测试输入上短暂。

还可以应用基于搜索的软件工程技术，例如Evodroid[42]和Sapienz[43]，它采用遗传编程来发展生成的测试输入，或Stoat[56]，它构成了随机模型并使用MCMC[8]指导这一代。此外，一些研究人员建议利用机器学习来引导输入生成[12,27,34]。此外，一些工作利用符号或Concolic执行来系统地生成测试输入，用于最大化分支覆盖，包括SIG-DROID[47]，Jensen等人提出的技术。[30]，合成[17]和Droidpf[7]。现有的基于搜索技术几乎缩放到大型应用程序。

最后，Coffodroid并不是第一个介绍Android应用程序测试中的组合概念。但是，现有的组合基础策略[1,48]关切的是单个事件的组合，因此无法生成（和有意义）的测试输入。

总之，所有现有技术都在为实际应用产生（且有意义）的测试输入，这对于体现深度应用态并揭示许多非活动错误至关重要。现有技术的局限性激励了组合的设计。

Android应用程序的半自动测试输入生成。提出了一些技术利用人类智能来提高生成的测试输入的质量。例如，偏振[44]从基于人群的测试中提取常见事件序列以增强SAPIENZ。App流程[29]记录人类提供的短事件序列，并利用机器学习来合成长事件序列。此外，UGA[38]扩展了探索应用状态空间的骨架的手动事件序列。虽然能够利用人类智能，但极化和UGA无法控制提取的手动事件序列的质量。另一方面，AppFlow缺乏有效机制，用于重用测试中的事件序列。

Android应用程序的域特定测试输入生成。某些技术旨在为某些测试域生成测试输入或表现出某种错误。例如，EoDroid[62]利用符号执行来为测试应用程序的WebViews生成输入，而Snowdrop[69]则旨在测试应用程序的后台服务。Apecher[16]和AATT+[36,57]在Android应用程序中生成测试输入以清单潜在的并发错误。此外，提出了一些技术来检测Android应用中的能量低效率，例如Greendroid[40]及其延伸[37,39,58]

这些技术通常与结合的正交。它们可以通过Combodroid产生的高质量测试输入受益。

用于GUI /

Web测试的测试输入生成。某些技术利用迭代GUI探索或程序分析来为GUI /

Web测试生成测试输入。有些工作[2,18-21,45,66,67]迭代地观察现有测试输入的执行，提取额外的知识（例如，精细模型），并衍生新的测试输入。例如，Nguyen等人。

[49]提出OEM

*范例，它在执行现有测试期间自动识别新的测试输入，扩展当前不完整的GUI事件模型，并基于当前执行迹线生成其他测试输入。这种迭代过程类似于Coffodroid。然而，这些技术提取的知识主要来自GUI转换的观察，并且诸如数据依赖性的测试输入之间的其他关系通常被忽略。另一方面，某些技术利用程序代码的静态分析来查找事件之间的数据依赖性，从而生成有效的测试输入[4,9,14,15,50]。但是，这些技术不能直接应用于测试Android应用程序，因为Android应用程序是基于障碍控制/数据流的组成部分，并且经常调用特定于Android的API以访问共享数据，例如，访问共享数据。sharedpreference.getBoolean。

相比之下，Cobicodroid从GUI转换和数据依赖性测试下的应用程序的知识，并利用具有Androidspecific API建模的执行跟踪的轻量级静态分析，以推断取决于输入之间的关系。

7 CONCLUSION AND FUTURE WORK

利用长期，有意义和有效的测试输入的洞察通常是短事件序列的串联，用于表现出特定的应用程序功能，提出了android应用测试输入生成问题的结构框架，分解为反馈循环案例生成和用例组合。评估结果令人鼓舞。完全自动的Combodroid α 平均覆盖的代码比最佳现有技术更多4.6%，并揭示了四个先前未知的错误。具有少人援助，半自动组合 β 与人类检测专家达到了可比覆盖率（平均水平仅3.2%）。CoffoDroid揭示了新的研究方向，以获得高质量的测试输入，全自动或人类援助。基于这种概念校验原型，可以在未来的组织增强中应用各种技术。有希望的研究包括利用机器学习在使用案例挖掘，人群源使用案例采集，以及模型检查组合。

ACKNOWLEDGMENTS

这项工作是由国家自然科学基金会（赠款#61690204，#61932021和#61802165）的支持。作者还要感谢江苏省新软件技术和工业化协作创新中心的支持。Yanyan Jiang (jyy@nju.edu.cn) 和chang xu (changxu@nju.edu.cn) 是相应的作者。

REFERENCES

[1] David Adamo, Dmitry Nurmuradov, Shradha Piparia, and Renée Bryce. 2018. Combinatorial-based event sequence testing of Android applications. *Information and Software Technology* 99 (2018), 98–117.

[2] Pekka Aho, Matias Suarez, Teemu Kanstrén and Atif M Memon. 2014. 墨菲工具：利用提取的GUI模型进行工业软件测试。2014年IEEE软件测试，验证和验证研讨会国际会议。IEEE, 343–348.

[3] Domenico Amalfitano, Anna Rita Fasolino, Porfirio Tramontana, Bryan Dzong Ta and Atif M Memon. 2014. Mobiguitar: 基于自动模型的移动应用程序测试。IEEE软件32,5 (2014), 53–59.

[4] Stephan Arlt, Andreas Podelski, Cristiano Bertolini, Martin Schäf, Ishan Banerjee and Atif M Memon. 2012. GUI测试的轻质静态分析。2012年IEEE 23经济软件可靠性工程国际研讨会。IEEE, 301–310. [5] Tanzirul Azim and Iulian Neamtii. 2013年. Android应用程序测试的有针对性的和深度探索。在2013年ACM Sigplan国际面向对象编程系统中的讨论会议的过程中的语言和应用程序。641–660.

[6] 年轻阔百克和斗湾裴。2016.使用多级GUI比较标准的基于自动模型的Android GUI测试。在第31届IEEE / ACM自动化软件工程国际会议上的诉讼程序中。238–249. [7] 广东白, 泉震叶, 永正吴, 黑景, 君孙, 杨柳, 金松洞, 威廉风光。2017.迈向模型检查Android应用程序。软件工程中的IEEE交易44,6 (2017), 595–612. [8] Steve Brooks, Andrew Gelman, Galin Jones and 小季萌。2011.手册

马尔可夫链蒙特卡罗。CRC压力机。

[9] 林成, Zijiang Yang和Chao Wang. 2017. GUI测试序列的系统减少。2017年32次IEEE / ACM自动化软件工程国际会议。IEEE, 849–860.

[10] Wontae Choi, George Neecula and Koushik Sen. 2013. Android应用程序的引导GUI测试具有最小的重启和近似学习。ACM Sigplan注意事项目48,10 (2013), 623–640.

[11] Shauvik Roy Choudhary, Alessandra Gorla and Alessandro Orso. 2015. Android的自动测试输入生成：我们是否存在？(e)。2015年30日IEEE / ACM自动化软件工程国际会议。IEEE, 429–440.

[12] 基督教德科特, 纳坦尼尔P Borges JR, Andreas Zeller. 2019年. 学习用户界面元素交互。在第28届ACM Sigsoft国际软件测试和分析研讨会上的诉讼程序中。296–306.

[13] 烟灰开发人员。2019.烟灰。从<https://github.com/>检索2019年6月29日

[14] Bernhard Dorninger, Josef Pichler and Albin Kern. 2015年. 利用静态分析从工业用户界面中提取知识提取。2015年IEEE软件维护和进化国际会议。IEEE, 497–500.

[15] Sebastian Elbaum, Srikanth Karre and Gregg Rothermel. 2003. 使用用户会话数据改进Web应用程序测试。在第25次国际软件工程召开, 2003年. 讨论者。IEEE, 49–59.

[16] 玲玲粉丝, 婷苏, 陈辰, 国湖萌, 杨柳, 李华旭和吉古普。2018. 有效地表现在Android应用程序中的异步编程错误。在第33届ACM / IEEE自动化软件工程国际会议的议程中。486–497.

[17] 翔高, 殷成璋, 甄涓和阿比克罗伊德夫利。2018. 通过合成符号执行的Android测试。在第33届ACM / IEEE自动化软件工程国际会议的议程中。419–429.

[18] 泽宇高, 镇宇陈, 云霄邵和艾夫M克。2015. SITAR: GUI测试脚本修复。软件工程中的IEEE事务42,2 (2015), 170–186. [19] 宙宝高, 春荣芳, 和艾夫M克。2015. 推动GUI回归测试中的自动化限制。2015年IEEE第26届软件可靠性工程国际研讨会。IEEE, 565–575.

[20] Ceren ahinGebizli, Abdulhadi Kerkic, 以及HasanSzer. 2018年通过风险驱动的基于模型的测试提高测试效率。系统与软件144 (2018), 356–365.

[21] CEREN SAHIN GEBIZLI, HASANSZER and ALI ZERERCAN. 2016年. 基于模型测试模型的连续改进, 提高系统测试效果。2016年IEEE第九次软件测试, 验证和验证研讨会会议。IEEE, 263–268.

[22] 谷歌。2019. Android调试桥 (adb)。从[HTTPS](https://developer.android.com/docs/)检索2019年6月29日:

[23] 谷歌。2019. Android文件。从<https://developer.android.com/docs/>检索2019年6月29日

[24] 谷歌。2019. Android UI Automator。从<https://developer.android.com/learn/testing/#uiautomator>检索到2019年6月29日2019年6月29日

[25] 谷歌。2019. Getevent。从[HTTPS://source.android.com/docs/testing](https://source.android.com/docs/testing)检索2019年6月29日的检索。COM / 设备/输入/ GETEVENT

[26] 谷歌。2019. UI / 申请锻炼者猴子。回收2019年6月29日

[27] Tianxiao Gu, Chun Cao, Tianchi Liu, Chengnian Sun, Jing Deng, Xiaoxing Ma, and Jian Liu. 2017. Aindroid: Activity-insulated multi-level automated testing for android applications. In 2017 IEEE International Conference on Software Maintenance and Evolution. IEEE, 103–114.

[28] Tianxiao Gu, Chengnian Sun, Xiaoxing Ma, Chun Cao, Chang Xu, Yuan Yao, Qirun Zhang, Jian Lu, and Zhendong Su. 2019. Practical GUI testing of Android applications via model abstraction and refinement. In 2019 IEEE/ACM 41st International Conference on Software Engineering. IEEE, 269–280.

[29] 帮派, 临主朱和俊峰杨。2018. Appflow: 使用机器学习合成强大, 可重复使用的UI测试。在2018年第26届ACM关于软件工程基础上的欧洲软件工程会议和研讨会上的议程。269–282.

[30] Casper S Jensen, Mukul Prasad, Anders Miller. 2013. 具有目标事件序列生成的自动化测试。在2013年关于软件测试和分析研讨会的诉讼程序中。67–77.

[31] Robert M Keller. 1976. 正式核实并计划。安排。ACM.

[32] Mountainminds GmbH & Co.KG和贡献者。2019. Jacoco Java代码覆盖率。从<https://www.jacoco.org/jacoco/trunk/index.html>检索到2019年的Appet 7,2019

[33] 平凡孔, 李立, 君高, 奎利, 托伐韦·贝斯迪夫, 以及雅克克莱茵。2018. Android应用程序的自动化测试: 系统文献综述。可靠性68,1 (2018), 45–66的IEEE交易。

[34] Yavuz Koroglu, Alper Sen, Ozlem Muslu, Yunus Mete, Ceyda Ulker, Tolga Tanirverdi and Yunus Donmez. 2018. QBE: 基于QLearning的Android应用探索。2018年IEEE第11届软件测试, 验证和验证国际会议。IEEE, 105–115.

[35] Kevin J Lang, Barak A Pearlmutter and Rodney Price. 1998. Abbadango一个DFA学习竞赛的结果和一种新的证据驱动状态合并算法。在语法推理的国际家族中。Springer, 1–12. [36] 齐伟李, 延安江, 天孝, 常旭, 君马, 小兴马, 建鲁。2016. 有效地表现在Android应用程序中的并发错误。2016年第23届亚太软件工程大会。IEEE, 209–216.

[37] Qiwei Li, Chang Xu, Yepang Liu, Chun Cao, Xiaoxing Ma, and Jian Liu. 2017. CyanDroid: stable and effective energy inefficiency diagnosis for Android apps. Science China Information Sciences 60, 1 (2017), 012104.

[38] Xiujiang Li, Yanyan Jiang, Yepang Liu, Chang Xu, Xiaoxing Ma, and Jian Lu. 2014. User guided automation for testing mobile apps. In 2014 21st Asia-Pacific Software Engineering Conference, Vol. 1. IEEE, 27–34.

[39] 易刘, 珏王, 常徐, 小兴马, 剑吕。2018. Navdyroid: Android应用程序的能源效率效率诊断的有效工具。科学中国信息科学61,5 (2018), 050103.

[40] Yepang Liu, Chang Xu, Shing-Chi Cheung and Jian Liu. 2014. Greendroid: 自动诊断智能手机应用的能源低效率。软件工程的IEEE事务40,9 (2014), 911–940.

[41] Aravind机械, Rohan Tahiliani and Mayur Naik. 2013. Dynodroid: Android应用程序的输入生成系统。在2013年第9届软件工程基金会联席会议的议程中。224–234.

[42] 利雅得Mahmood, Nariman Mirzaei and Sam Malek. 2014. Evodroid: Android应用程序的分段演进测试。在第22届ACM Sigsoft国际软件工程基金会研讨会上的议程中。599–609. [43] Ke Mao, Mark Harman and Yue Jia. 2016. SAPIENZ: Android应用程序的多目标自动化测试。在第25国际软件测试和分析研讨会的诉讼程序中。94–105.

[44] Ke Mao, Mark Harman and Yue Jia. 2017年. 人群智能增强了自动移动测试。2017年32次IEEE / ACM自动化软件工程国际会议。IEEE, 16–26.

[45] ATIF MEMON, ISHAN BANERJEE和Adithya Nagarajan. 2003. GUI撕裂: 用于测试的图形用户界面的逆向工程。在逆向工程的第10次工作会议中, 2003年. WCRE 2003.诉讼程序。CITSEER, 260–269. [46] 国湖梦, 云兴薛, Chandramohan Mahinthan, Annamalai Narayanan, Yannamalai Narayanan, Yang Liu, Jie Zhang and Tieming Chen. 2016. MyStique: 审计Android恶意软件以审计反恶意软件工具。在亚洲计算机和通信安全亚洲会议上的第11届ACM的会议记录中。365–376.

[47] Nariman Mirzaei, Hamid Bagheri, Riyadh Mahmood and Sam Malek. 2015. Sigdroid: Android应用程序自动化系统输入生成。2015年IEEE第26届软件可靠性工程国际研讨会。IEEE, 461–471. [48] Nariman Mirzaei, Joshua Garcia, Hamid Bagheri, Alireza Sadeghi and Sam Malek. 2016年. 减少了Android应用的GUI测试中的组织。2016年IEEE / ACM第38届软件工程国际会议。IEEE, 559–570. [49] Bao Nguyen and Atif M Memon. 2014年. 遵守模型练习*范例, 以测试具有未确定输入空间的事件驱动系统。软件工程中的IEEE事务40,3 (2014), 216–234.

[50] Jacinto Reis and Alexandre Mota. 2018. 促使修剪的探索性测试GUI模型。通知。过程。吧。133 (2018), 49–55.

[51] 问题128呼叫。回收2019年6月29日 <https://github.com/f>

[52]问题报告。2019年。关于AnyMemo的问题480。从[https://github](https://github.com)

[53]问题报告。2019。
AnyMemo的问题503。从<https://github.com/hocloWorld1/anymemo/issues/503>检索2019年6月29日

[54]问题报告。2019。
AARD2的第90期。从<https://github.com/itkach/aard2-android/issues/90>中检索2019年6月29日

[55]魏松, 湘兴钱, 杰夫黄。2017。
Ehbdroid: 超越Android应用的GUI测试。2017年32次IEEE /
ACM自动化软件工程国际会议。IEEE, 27–37。

[56]婷苏, 国湖萌, 玉婷陈, 柯武, 威梅杨, 姚瑶, 古煌蒲, 杨柳和苏颂。2017。
android应用程序的基于随机模型的GUI测试。在2017年第11届关于软件工程基金会的第11
次联席会议的议程中。245–256。

[57] Yue Wang, Yanyan Jiang, Chang Xu, Qiwei Li, Tianxiao Gu, Jun Ma, Xiaoxing Ma, and Jian
Lu. 2018. AATT+: Effectively manifesting concurrency bugs in Android apps. Science of Computer
Programming 163 (2018), 1–18.

[58]珏王, 益邦刘, 张旭, 小兴马, 建鲁。2016年。E-Greendroid: Android应
用的有效能量低效率分析。在第8届亚太地区互联网仪器研讨会上。71–80。

[59]文字王, 邓丰李, 魏阳, 玉瑞曹, 张, 玉阳邓, 陶谢。2
018.工业案例中Android试验工具的实证研究。在第33届ACM
/ IEEE自动化软件工程国际会议的议程中。738–748。

[60] Renato Werneck, Joao Serubal和Arlindo da
Conceicao。2000.找到最小拥塞跨越树木。实验算法5 (2000), 11–ES。[61]湘
宇吴, 延安江, 张旭, 春曹, 小兴马, 建璐。2016.通过引导姿态事件生成测试
Android应用程序。2016年第23届亚太软件工程大会。IEEE, 201–208。

[62]广良杨和杰夫黄。2018.在Android混合应用程序中自动生成事件
导向的漏洞利用。在proc. 网络与分布式系统安全研讨会。

[63]坤杨, 建威卓格, 永科王, 鲁汇周和海新段。2014。
IntentFuzzer: 检测Android应用程序的能力泄漏。在第9届AC
M信息, 计算机和通信安全性研讨会的议程中。531–536。

[64]汇辉, 绍伊宁成, 兰博张, 樊江。2013。
DroidFuzzer: 使用意图过滤器标记模糊Android应用程序。在移动计算和多媒体进步国际会
议的诉讼中。68–74。

[65]赵春辉, 韩琳·陆, 春元陈, 克泉朗, 和施坤黄。2014。
Craxdroid: 通过选择性符号执行, 自动Android系统测试。2014年IE
EE第八届软件安全和可靠性伴侣国际会议。IEEE, 140–148。

[66]迅大和丈夫M克。2008.交替的GUI测试生成和执行。在测试中: 学术和工
业会议 – 实践和研究技术 (TAIC第2008号)。IEEE, 23–32。

[67]荀元和丈夫M克。2009.使用GUI运行时状态反馈生成基于事件的基于序列
的测试用例。软件工程中的IEEE交易36,1 (2009), 81–95。

[68] Shengtao Yue, Weizan Feng, Jun Ma, Yanyan Jiang, Xianping Tao, Chang Xu, and Jian Lu.
2017. RepDroid: an automated tool for Android application repackaging detection. In 2017
IEEE/ACM 25th International Conference on Program Comprehension. IEEE, 132–142.

[69]李丽娜张, 千豪迈克梁, 云新刘, 晨康。2017.系统地测试移动应用的背景
服务。2017年32次IEEE / ACM自动化软件工程国际会议。IEEE, 4–15。