REP FLOW

基于对话系统引导的
移动应用自动化测试结果复现技术

211250117 周博龄
211250062 吴陈添
211830015 袁　晨
211250165 刘尧力

2023 年秋 软件测试

南京大学

2024 年 1 月 5 日

目录

1. 项目简介

项目地址: https://github.com/wuc9521/rep-flow
TODO: @zbl

1.1. 项目背景

本项目设计了一种基于对话系统引导的自动化测试结果复现技术，通过对自动化测试结果的有效分析与相关测试知识库的指导，对话系统可根据测试人员当前复现情况进行有效的引导，使其更高效地完成自动化测试结果复现的工作。

1.2. 项目结构

```
rep-flow/
├── README.md
├── Makefile: 项目的主要配置文件.
├── data/
│   ├── README.md
│   ├── apk: 用于复现 bug 的 apk 文件
│   ├── corpus/: 用于对话系统的语料库.
│   ├── guidance/: 存放复现需要的图片，以 bug 编号分类.
│   ├── list/: 存放复现 bug 的动作列表，以 bug 编号命名.
│   └── state/: 存放用户当前状态的截图，以时间戳命名.
├── log/: 存放项目运行过程中生成的后端日志和脚本日志.
├── script/
│   ├── README.md
│   ├── __init__.py
│   ├── config.json: 设备的配置文件.
│   └── main.py: 用于获取用户当前状态的脚本.
├── model/.
│   ├── __init__.py
│   ├── common.py: 用于预处理图片.
│   ├── hist.py: 三直方图算法模型实现与测试.
│   ├── ssim.py: 结构相似性算法模型实现与测试.
│   └── process.py: 分析图片相似度的主程序.
├── static/: 前端的静态文件.
├── index.html: 前端的主页.
├── app.py: 后端的主要逻辑.
└── test/
    ├── README.md
    ├── monkey.py: 使用 monkey 工具对目标应用执行随机测试
    ├── script/: 随机测试复现和过程截图获取
    └── linkedlist.py: 负责数据格式转换，测试结果以链表形式存储备用
```

- `requirements.txt`: 涉及到的 Python 依赖
- `package.json`: 涉及到的 Node 依赖
- `utils/`: 涉及到的工具文件

2. 功能模块说明

从功能设计的角度, 我们的工具按执行顺序主要分为自动化测试执行、自动化测试结果分析、复现引导。

2.1. 自动化测试执行

在自动化测试执行阶段，我们共做了两种尝试。首先在 Appium 框架的基础上应用 monkey 工具对目标安卓应用进行随机测试，通过调整随机动作的比例和间隔尝试触发 bug，将成功触发 bug 的 log 保存为文件，每个动作的参数包括动作类型（按下、抬起、翻转、截图等）和作用坐标。

由于 monkey 工具不支持在运行的同时保存每次动作前后的截图状态，我们额外使用/test/script 中代码自定义动作复现被 monkey 触发的 bug，以获取行动链每个节点的截图。自动化测试执行的结果再经过下一步“测试结果分析”处理，以备引导复现使用。

2.2. 自动化测试结果分析

每次 monkey 运行发现一个 bug，都会将发现错误的过程截图保存到一个文件夹下，但是 monkey 在运行过程中是随机操作的，因此会生成一个非常冗长的操作序列，我们很难确定最终的 bug 是由哪些序列引起的，在项目中，我们采用了一些简单的手段进行处理。

- 邻近图片去重：monkey 可能会随意点击而无法触发任何交互效果，我们使用图片相似度分析的手段寻找相邻且高度相似的图片，仅保留一张
- 寻找开始与结束序列：无论 monkey 如何执行序列，开始序列的第一张图应当是手机的 Home 界面，我们将 Home 界面作为一段执行序列入口，找到两个邻近并且包含了引发 crash 的操作截图的入口，仅保留这两个入口间的截图即可
- 我们没有设计将特定组件识别并且展示给用户的功能，为了方便用户交互，我们围绕 monkey 给出的 location 参数以一定的半径画一个圈，以此来提示用户点击、滑动等操作

但是经过这些处理后的效果仍然不理想，最后我们还是对 bug 图像序列进行了一些手动修正。

2.3. 复现引导

2.3.1. 对话系统的实现

我们设计了一个轻量的对话系统. 具体来说, 后端使用 `Python Flask` 框架, 前端使用纯 `html` 与 `css` 实现. 我们选取了 `Python SpaCy` 库中的 `en_core_web_sm` 小型语言模型用作自然语言处理.

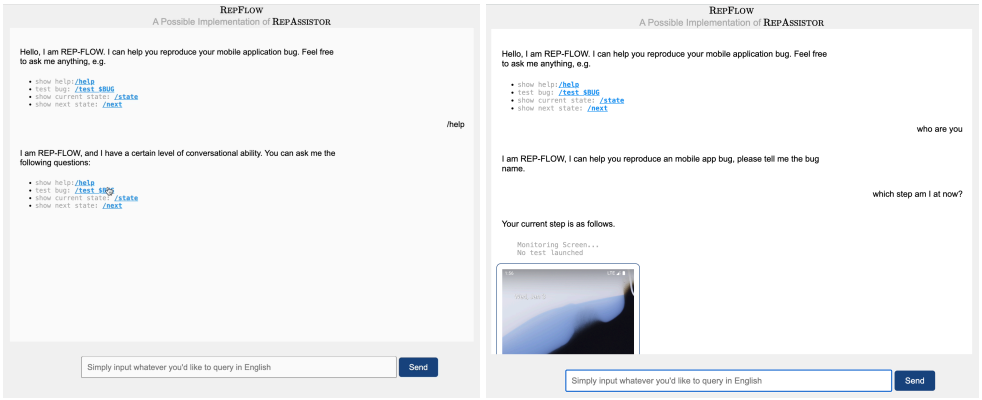
我们考虑到当前的场景是高度专业化的, 因此对话系统的功能设计也是针对性的. 出于这样的想法, 我们并没有接入更大的语言模型, 而是在本地规定了一些常见的问题作为语料库, 使用 SpaCy 库的 similarity() 函数作为相似度的度量. 基本的想法是, 如果用户提出了一个问题, 那么我们就在语料库中寻找与之相似的问题, 并返回相应的答案. 对应的代码如下:

```
question = DEFAULT_RESPONSE_FLAG
for question_, answer in qa.items():
    if doc.similarity(nlp(question_)) > doc.similarity(nlp(question)):
        response = answer
        question = question_
if response == doc.similarity(nlp(question)) < 0.7:
    app.logger.warning(f"User query: \"{query_text}\" - No answer found")
    ......
```

在实现的过程中, 我们主要将用户的问题分成了如下的几类, 并给出相应的回答:

- [ERROR]: “I’m sorry, I can’t understand you.”
- [GREETING]: “Hello, what can I help you?”
- [NEXT]: “You are supposed to reach this step as shown in the following screenshot.”
- [CURRENT-STATE]: “Your current step is as follows.”
- [HELP]: “I am REP-FLOW, and I have a certain level of conversational ability. You can ask me the following questions..”
- [SORRY]: “I’m sorry, I’m a simple chatbot, which means that I can only help you reproduce the bug according to the recorded steps. If you want to reproduce the bug in other ways, please try again.”
- [TEST]: “Sure thing! You can try reproducing the bug yourself, and I would monitor your actions and record the steps. If you are not sure what the next step is at a certain point, you can ask me. e.g. ’What’s the next Step?”
- [ID-MISSING]: “Please tell me the bug number first, e.g. /test 114514.”
- etc.

这样的设计使得我们的对话系统即满足领域专业化的要求, 又能够在一定程度上满足用户的需求. 例如, 如果用户在复现的过程中不知道下一步是什么, 那么他可以根据指示点击 /next 指令 (如图1a), 也可以直接问对话系统下一步是什么 (如图1b).



(a) [对话系统] 提示　　(b) [对话系统] 对话

2.3.2. 状态提取

我们使用 `Appium` 来获取用户屏幕的状态. 当对话系统启动时, 启动个新的进程来运行 `Appium` 脚本. 我们规定, 如果:

1. 当前页面的 `xml` 结构和上一个时刻的 `xml` 结构不同, 那么就认为用户进行了一次操作; 或者
2. 当前页面的任何一个 `Button` 元素被点击 (通过 XPath 来实现); 或者
3. 当前页面还没有被截图过.

那么就截取图片并且存储在本地, 当前端请求的时候发送给前端. 具体来说, 我们使用如下的 `while` 循环来实现:

```
try:
    while True:
        current_page_source = get_current_page_source(driver)
        if current_page_source != previous_page_source or
    is_button_pressed(driver):
            screenshot_path = os.path.join(STATE_DIR, f"{time.time()}.png
    ")
            capture_screenshot(driver, screenshot_path)
            previous_page_source = current_page_source
        time.sleep(0.01)
except KeyboardInterrupt:
    print("Exiting the loop.")
finally:
    driver.quit()
```

2.3.3. 图片相似度分析

为了确认用户在 bug 的复现流程中处于哪一步, 我们需要对当前用户截屏和 bug 复现截屏流进行图片相似度分析, 因此我们使用了以下两种相似度分析算法模型.

1. 三直方图算法模型. 算法主要基于图像的颜色信息，通过比较图像的颜色直方图来评估它们之间的相似性. 代码如下:

```
    def calculate(image1, image2):
        hist1 = cv2.calcHist([image1], [0], None, [256], [0.0, 255.0])
        hist2 = cv2.calcHist([image2], [0], None, [256], [0.0, 255.0])
        degree = 0
        for i in range(len(hist1)):
            if hist1[i] != hist2[i]:
                degree = degree + \
                    (1 - abs(hist1[i] - hist2[i]) / max(hist1[i], hist2[i
        ]))
            else:
                degree = degree + 1
        degree = degree / len(hist1)
        return degree

    def classify_hist_with_split(image1, image2, size=(1000, 2000)):
        image1 = cv2.resize(image1, size)
        image2 = cv2.resize(image2, size)
        sub_image1 = cv2.split(image1)
        sub_image2 = cv2.split(image2)
        sub_data = 0
        for im1, im2 in zip(sub_image1, sub_image2):
            sub_data += calculate(im1, im2)
        sub_data = sub_data / 3
        return sub_data
```

2. 结构相似性算法模型. 算法主要通过分别比较两个图像的亮度，对比度，结构，然后对这三个要素加权并用乘积表示. 代码如下：

```
image_user = io.imread(image_user_path)
gray_user = cv2.cvtColor(image_user, cv2.COLOR_BGR2GRAY)
for i in range(len(image_list)):
    (score_ssim, diff) = sk_cpt_ssim(gray_user, gray_list[i], full=
    True)
```

为了提升图片相似度分析的准确率, 我们采取加权平均的方式综合两种算法模型的结果, 当最大综合相似度 >0.65 时认为找到了用户所处的 bug 复现位置, 反之则认为用户脱离了正常的 bug 复现流程. 具体实现代码如下:

```
image_user = io.imread(image_user_path)
gray_user = cv2.cvtColor(image_user, cv2.COLOR_BGR2GRAY)
max_score = 0
max_similar = 0
for i in range(len(image_list)):
    score_hist = classify_hist_with_split(image_user, image_list[i])
    (score_ssim, diff) = sk_cpt_ssim(gray_user, gray_list[i], full=True)
    average_score = (3*score_hist+2*score_ssim)/5
    if average_score > max_score:
        max_score = average_score
        max_similar = i
if max_score < 0.65:
     return None
return max_similar, max_score
```

我们注意到, 结构相似性算法模型的时间开销明显大于三直方图算法模型, 所以提供了

另一个只使用三直方图算法模型的函数接口，该函数的运行速度是上一种函数的十倍左右。当最大相似度 >0.7 时认为找到了用户所处的 bug 复现位置，反之则认为用户脱离了正常的 bug 复现流程.

当在对图片相似度分析的准确度要求不高的情况下，我们选择第二种函数，以提高对话系统的整体反应速度. 具体实现代码如下:

```
image_user = io.imread(image_user_path)
max_score = 0
max_similar = 0
for i in range(len(image_list)):
    score = classify_hist_with_split(image_user, image_list[i])
    if score > max_score:
        max_score = score
        max_similar = i
if max_score < 0.7:
    return None
return max_similar, max_score
```

3. 运行、交互说明

3.1. 工具基本信息说明

安卓版本: Android 13, Pixel 4 XL
从运行和交互的角度，主要分为自动测试和复现引导两个模块，其中两个模块统一版本，但相互独立，可以分别运行，分别对应测试人员和复现人员的需求。

3.2. 测试模块

测试人员首先需要保持 Appium server 启动

```
appium
```

然后运行虚拟机或者连接手机，再新建终端运行/test 路径下的 monkey.py

```
python monkey.py
```

如果遇到运行问题，参考 README 中的指引，可能需要更改 python client 的版本。如果要更改随机测试的目标或参数需要更改 monkey.py 中的相应参数

3.3. 复现引导模块

我们为项目配置了完整的 `Makefile`. 为了运行项目，只需要在项目根目录下运行 `make` 就可以看到详细的指示.

```
[~/rep-flow]$ make
Running on Darwin

Usage: make [target]

Available targets:
  install : install dependencies.
  run     : run the app.
  clean   : clean up the logs and figures.
  stop    : stop the app.
  boot    : boot the emulator.!!! modify the emulator name in Makefile.
  reload  : reload the appium server.

Normally, run make install first, then run make run to start the app.
```

一般来说, 用户需要先接上手机或者运行虚拟机, 然后在项目根目录下运行

```
make install
```

来安装需要的 Python 和 Node.js 依赖. 然后运行

```
make run
```

来启动对话系统, 模型和检测脚本.

```
[~/rep-flow]$ make run
Cleaning data/state...
Cleaning log...
Cleaned up.
[INFO] 1 deviced detected
App is running...
Script is running...
Please open http://localhost:5000
```

目前我们只支持 Android 平台, 并且我们默认使用的是 Pixel 4 XL 的模拟器. 如果你想要使用其他的设备, 请修改 `script/config.json` 中的相应变量.

参考文献

[1] Xin Li et al. “Towards Effective Bug Reproduction for Mobile Applications”. In: *2023 10th International Conference on Dependable Systems and Their Applications (DSA)*. Tokyo, Japan: IEEE, Aug. 2023, pp. 114–125. ISBN: 9798350304770. DOI: 10.1109/DSA59317.2023.00024.