Crowd Intelligence Enhances Automated Mobile Testing

Ke Mao*, Mark Harman*, and Yue Jia*
Facebook London, Facebook, 10 Brock Street, London, NW1 3FG, UK
CREST, University College London, Malet Place, London, WC1E 6BT, UK
{kemao, markharman, yuej}@fb.com

***Abstract*—We show that information extracted from crowd-based testing can enhance automated mobile testing. We introduce POLARIZ, which generates replicable test scripts from crowd-based testing, extracting cross-app 'motif' events: automatically-inferred reusable higher-level event sequences composed of lower-level observed event actions. Our empirical study used 434 crowd workers from Mechanical Turk to perform 1,350 testing tasks on 9 popular Google Play apps, each with at least 1 million user installs. The findings reveal that the crowd was able to achieve 60.5% unique activity coverage and proved to be complementary to automated search-based testing in 5 out of the 9 subjects studied. Our leave-one-out evaluation demonstrates that coverage attainment can be improved (6 out of 9 cases, with no disimprovement on the remaining 3) by combining crowd-based and search-based testing.**

***Index Terms*—Crowdsourced Software Engineering, Mobile App Testing, Test Generation**

I. INTRODUCTION

There has been much recent progress in automated testing [1], [2], with recent advances in automated testing of mobile apps [3]. However, automated test data generation techniques lack domain knowledge, and may generate either unrealistic test cases or fail to find test cases that explore aspects of functionality that matter to users [4]. A recent study of open source Android apps with relatively simple user flows [3] reported that current state-of-art automated mobile testing techniques achieve only approximately 50% statement coverage.

Fortunately, Linares-Vásquez et al. [5] recently showed how app execution usage data can be mined for valuable insights, while Moran et al. subsequently introduced CrashScope [6] which supports the discovery of app crashes and their replication. There has also been considerable recent interest in the possibilities of crowdsourcing as a means of collecting such usage data in a cost-effective manner. This recent work suggests that data mining and extraction, perhaps from crowd sourced usage, might discover useful cross-app patterns that improve automated app testing performance.

We explore the complementarity between automated machine-generated tests and human (crowd-generated) tests. We specifically focus on the ability of crowd-based tests to assist the state-of-the-art search based testing tool, Sapienz. An open source research prototype of Sapienz [7] was released in 2016, and the technology that underpins it has been under development at Facebook since February 2017. In this paper we use the open source version of Sapienz to facilitate replication. The Sapienz approach to search based testing is well-suited to augmentation with crowd-based tests due to Sapienz' concept of a motif gene: a sequence of low-level events that has a (context-sensitive) meaning to the app's users, thereby denoting an 'atomic' event (to users), although appearing to be a non-atomic event sequence (to the device and any testing approach that lacks the necessary context-awareness).

We show that these strands of work on mining, crowdsourcing and automated test generation can be combined in a mutually-complementary hybrid. Our hybrid uses automated test generation to explore the search space of test cases, informed by data mined from crowdsourced tests to identify motif genes. To do this, we introduce a crowd-based approach, called POLARIZ, which collects and analyses test inputs from a crowd call to non-technical users with no specific software testing expertise or experience. That is, the call is open to any crowd workers to participate, whether or not they have testing expertise. However, since it is an open call (in the spirit of crowdsourcing) we cannot guarantee that we do *not* recruit any crowd workers with testing expertise.

POLARIZ uses a platform with a mobile device infrastructure, remote device control and screen streaming, automated subject distribution, permission control and crowd trace collection. With this approach, a non-professional crowd from the general public (such as those from Amazon Mechanical Turk) can contribute to mobile testing from anywhere with any clients with a web browser (e.g., desktop PC, Android, iOS or Windows Phone mobile devices).

Following Linares-Vásquez et al. [5], we introduce a novel data mining algorithm to extract 'motif' event sequences; sequences composed of lower-level events that our approach infers may denote higher-level atomic units, thereby providing one source of guidance for automated testing. We define a 'motif' event sequence (or 'motif pattern') as a common user interaction pattern that is learned from some apps, and can be subsequently generalised to other apps, such that the motif sequence can play the role of a higher-level atomic event that can be re-used to assist automated mobile testing.

*This research forms part of the PhD work of Ke Mao, the lead author, supervised by Yue Jia and Mark Harman while all three were at University College London. Dr. Mao, Dr. Jia and Prof Harman moved to Facebook full time in February 2017 and Dr. Jia and Prof. Harman also retain part time positions at UCL.

A ‘motif pattern’ may occur multiple times (there may be many instances of a motif pattern), each occurrence of which we refer to as ‘motif events’. Our approach thus bridges the gap between automated mobile test input generation techniques and human domain/context awareness, using non-professional crowd testers.

The primary contribution of our work is the scientific evidence that motifs extracted from crowdsourced tests can complement and extend state-of-the-art automated test generation. More specifically, the contributions of our work are as follows:

1) We introduce and implement the POLARIZ approach for harnessing crowd intelligence to support mobile testing. Using our implementation, we report the results of the first empirical study of crowdsourcing for mobile test automation. We posted 1,350 tasks on Amazon Mechanical Turk to test 9 popular Google Play apps, each with at least 1 million user installs. The crowd was able to attain 60.5% overall (unique) activity coverage.
2) We compare the results of app activities covered by the crowd with those by the automated search based Android testing tool, SAPIENZ, revealing complementarity between the two. We chose SAPIENZ because it has recently [7] been shown to outperform other state-of-the-art and state-of-practice tools. Unsurprisingly, the crowd, imbued with its superior domain knowledge, achieved higher activity coverage on all but one subject (Google Translate, for which SAPIENZ produced slightly higher coverage). More importantly, for 5 the 9 subjects the two techniques exhibited complementary coverage, motivating our goal of combining them.
3) We introduce a motif-extraction algorithm and demonstrate its effectiveness, by showing that it can enhance SAPIENZ’ coverage. For 6 of the 9 subjects the coverage is improved by motif extraction, with the best case improvement increasing coverage obtained from 6 to 12 (of 27 possible) unique activities.

II. THE POLARIZ APPROACH

The POLARIZ approach is designed to tackle the two main challenges involved in harnessing the non-professional crowd to perform remote mobile testing, and further to learn from crowd intelligence embodied in the crowdsourced manually constructed tests. The first challenge requires an intermediary platform, able to harness a general public crowd to work on remote mobile testing tasks. The second challenge involves the representation and extraction of useful crowd intelligence.

Figure 1 depicts the high-level POLARIZ workflow. Three actors are involved in the workflow:

1) App developer/researcher who seeks mobile test automation enhancement;
2) Crowd workers/testers;
3) The intermediary platform (i.e., POLARIZ platform) on which the crowd works.

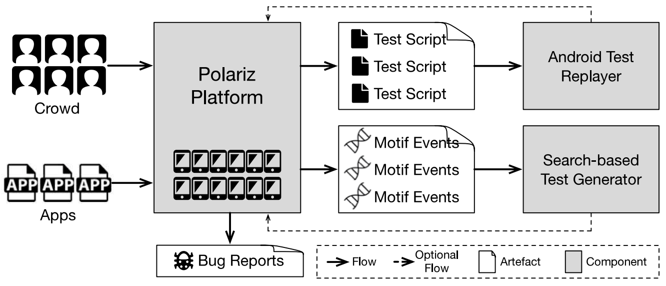

Fig. 1. Overall workflow of Polariz

POLARIZ uses its own device infrastructure; users do not execute apps on their own devices. This insulates the user from security issues, while insulating POLARIZ against Android device fragmentation. It also gives POLARIZ full control over real-time data collection and monitoring. However, it means that POLARIZ can only collect touch-screen events, not device-specific events such as GPS and accelerometer events. It also involves a latency (since testing activity occurs over the network), which we checked and report on. Fortunately, these results indicate that the latency is acceptable.

The outputs consist of three parts: First, the bug reports automatically generated during the crowd testing process; Second, the replicable test scripts generated based on crowd-sourced test manual traces, which can be replayed via an Android test replayer (such as RERAN [8]); Third, the automatically summarised motif events learned from crowd interaction traces, which can be further used to enhance existing search-based mobile test generators (such as SAPIENZ). Both the Android test replayer and test generator can remotely connect to POLARIZ’ mobile device infrastructure for test execution.

POLARIZ’ top level consists of its crowd testing platform (for manual trace collection), and its crowd motif extraction algorithm (for learning from crowd intelligence). Our platform uses crashing as an implicit oracle [9], automatically capturing crash-triggering stack traces, event sequences and witness videos using the existing SAPIENZ infrastructure [7].

A. The Polariz Platform

The platform is illustrated in Figure 2. Given a set of mobile apps under test, POLARIZ’ subject dispatcher component automatically instruments, assigns and installs each app on a device in its mobile device infrastructure. The screen streamer and device controller provide web services for controlling these devices. Crowd users simply access the remote devices which install apps via web browsers from any user platform.

Exposing our hosted mobile devices to the general public might raise security concerns, so POLARIZ has a permission control component that monitors crowd interactions, only permitting testing activities on the specified subjects. During the the crowd testing process, POLARIZ’ logging components such as a crash detector and trace collector automatically collect the information from which POLARIZ generates its reports.

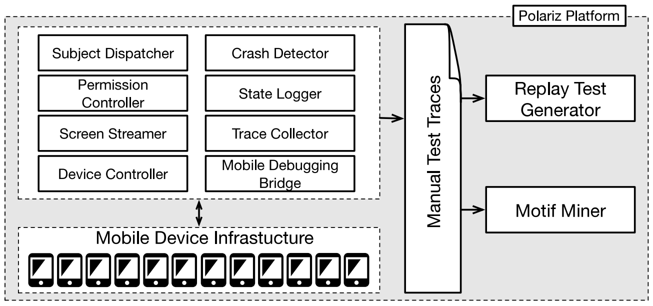

Fig. 2. Detailed components of Polariz platform

B. The Crowd Motif Extraction Algorithm

Our use of crowd 'motif' patterns stems from DNA sequence motif. According to D'Haeseleer [10], a DNA sequence motif is a short, over-represented pattern with an assumed biological function. The crowd motif extraction problem is to find a set of recurring substrings within a set of strings, which can be described as follows:
Given a set of N crowd-generated test event sequences $\mathcal{S} = \{S_1, \ldots, S_N\}$, each formed by events from an event set:

$$\mathrm{Y} = \{Swipe, Rotate, Flip, Pinch, Click_{ROI1}, Click_{ROI2}, \ldots, Press_{key1}, Press_{key2}, \ldots\} \quad (1)$$

the motif extraction problem is to find a set of instances $M = \{m_1, \ldots, m_n\} (n \leq N)$, where each m_i is a w-sized subsequence of sequences in $\mathcal{S}$ that maximises M's information content (IC$_M$) according to the equation:

$$\mathrm{IC}_M = \sum_{i=1}^{w} \sum_{y \in Y} p_{y,i} \log \frac{p_{y,i}}{\mathbb{B}_y}, \quad (2)$$

where $p_{y,i}$ is the probability of event y at position i in M, and $\mathbb{B}_y$ is the probability of event y in the background distribution. Thus, IC$_M$ computes the relative entropy of the event sequences in M, favouring those sub-sequences that are prevalent in the crowd's behaviour, yet are relatively less prevalent in the overall distribution. Our approach is inspired by DNA sequence motif discovery [11]–[13].

We use a genetic algorithm to extract crowd motif events for mobile testing. The adapted crowd motif extraction algorithm is listed in Algorithm 1. The algorithm extracts multiple motif patterns from a set of collected log-trace pairs. The log provides subject execution state information, such as transitions from one app activity to another, and the trace saves manual interactions that were used to trigger the app's state changes. The log and trace items are linked via their timestamps.

In order to learn from 'the wisdom of the crowd', Lines 2-3 extract the minimum trace that enables a transition from one activity to another. That is, there may exist many ways (sequences of events) through which the user interactions trigger the same app activity. We favour the 'minimum trace' that requires the fewest operations. The generated minimum transitional trace collection, $\mathcal{S}$, is represented as a list of strings, in which each string is a minimum trace that triggers a specific 'A to B' activity transition.

Algorithm 1: Crowd Motif Extraction Algorithm

```
1  Description: Find $m$ motif patterns from $n$ subject log-trace pairs.
   Input: A list of log-trace pairs $\mathcal{D}$ = [($L_1$, $T_1$), ($L_2$, $T_2$) ,..., ($L_n$, $T_n$)], where
          $L_i$ is the app execution state log and $T_i$ is the app event traces for the $i$th
          app; Number of motif patterns to find $m$.
   Output: A list of recurring motif patterns $R$ = [$r_1$, $r_2$, ..., $r_m$].
2  $R \leftarrow$ [], $S \leftarrow$ [];                          ▷ initialisation
   ▷ get the minimum operations to switch from one activity
     to another
3  for each ($L$, $T$) in $\mathcal{D}$ do
4      $S \leftarrow S \cup getMinimumActivityTransitionTrace((L, T))$;
   ▷ find $m$ motif patterns by evolving candidate motif
     substring locations
5  for $i$ in $range(0, m)$ do
6      generation $g \leftarrow 0$;
       ▷ for each individual generate random candidate motif
         locations in S
7      $P \leftarrow initialisePopulation(S)$;
8      evaluate $P$ by calculating $IC_M$ for each individual in $Q$;      ▷ see
         Equation 2
9      while $g < max\_generations$ do
10         $P' \leftarrow tournamentSelection(P)$;
11         $Q \leftarrow variation(P')$ ;    ▷ crossover and mutate motif
             locations and length
12         evaluate $Q$ by calculating $IC_M$ for each individual in $Q$;
13         $Q \leftarrow elitismSelection(Q, P)$;
14         $g \leftarrow g + 1$;
15         $P \leftarrow Q$;
16     $r \leftarrow getBestIndividual(P)$;     ▷ may contain zero or one
         motif location for $s \in S$
17     $R \leftarrow r \cup R$;
       ▷ excluding found motif substrings for next motif
         pattern
18     $S \leftarrow removeMotifSubstrings(S, r)$;
19 return $R$;
```

Lines 5-18 use a genetic algorithm to find multiple motif patterns. At each iteration, the algorithm finds a single motif pattern and excludes the matched motif substrings from S (Line 18). Each individual genetic algorithm population member represents a candidate motif pattern; a list of candidate motif locations in S. The individual's fitness is evaluated based on the information content score, as described by Equation 2.

The variation operator (Line 11) applies both crossover and mutation to manipulate the location and length of each motif substring. The best individuals (with highest information content score, i.e., their motif substrings are most conservative) are selected for the next generation. In this way our genetic algorithm uses elitism in its selection and retention. Evolution stops after a given maximum number of generations, saving the best individual found. The overall process repeats until all m motif patterns have been discovered.

Our implementation consists of the two top level components as described in Section II to produce the platform shown in Figure 2. POLARIZ implementation's mobile device infrastructure consists of 9 Nexus-7 tablets, connected to a host PC via a USB hub. We adapt the Android 'getevent' tool for trace recording, and use the RERAN [8] tool for trace replay: The 'getevent' tool captures a list of low-level Android events on-the-fly, saving to a script which is subsequently interpreted by the RERAN tool. For remote device control, we use the open sourced 'openstf' project[1], and we deployed the platform to a server with a proxy service to speed up global visits.

[1]https://github.com/openstf/stf

We use the SAPIENZ implementation obtained from the open source prototype, made available by the SAPIENZ research project [7]. When improving SAPIENZ, we integrate the learned motifs into SAPIENZ' MOTIFCORE component, which combines the motif patterns (which SAPIENZ calls 'motif genes') with atomic 'genes'. In order to learn the crowd motifs from the collected event traces, we implement the POLARIZ motif extraction component in Python, according to Algorithm 1. SAPIENZ can record and replay event sequences and uses the low-level Android monkey representation of events to go this. This low level representation is not immediately human-readable, but since our event sequences (and motifs extracted) are intended purely for machines this is not a problem.

III. EMPIRICAL EVALUATION OF POLARIZ

We want to investigate the usefulness of POLARIZ, both as a source of test data, garnered from an untrained and technically non-specific crowd, and also as a mechanism for augmenting existing automated techniques for test data generation. In this section we outline and motivate the four research questions that we choose to pose and answer in this paper:

RQ1: Demographics and behviour: Before investigating the nature of test cases generated and analysed from our crowd workers, we first report on the demographic diversity and behavioural characteristics of the crowd. We do this to support comparison and replication and subsequent study, which will exhibit inherent variability due to the nature of the crowd recruited for any such subsequent study.

RQ1.1: The demographic diversity of the crowd: RQ1.1 reports on the diversity of the crowd workers recruited in order to perform the testing activities in our study. In order for the crowd to denote an affective source of test data, which exhibits diversity, the crowd itself will need to be diverse. This diversity is important in order to ensure that the test cases explore app behaviour exercised by all of the types of users who may use the application under test. It would also be important for the motif extraction approach, because this needs to generalise from a set of instances, observed from crowd behaviour. If we lack diversity in the crowd, then there is the possibility that the motif extraction algorithm will overfit.

We report on the diversity of the crowd in terms of demographic distribution, gender, educational background, and prior experience, both with mobile applications in general, and software testing in particular. We also report on the level of returning workers; those who come back to complete further testing tasks that we set, having already tackled our previous testing tasks.

RQ1.2: The crowd's interest level: In order to be sufficiently motivated to tackle the testing tasks we set, the crowd needs to feel interested in these tasks. The tasks we set are not specifically related to testing, but simply involve using the applications under test. We survey the crowd for their self-assessed, level of interest, on a standard Lickert scale, in order to provide some initial evidence relating to the level of crowd interest.

RQ1.3: The crowd's response rates: We also investigate the behaviour of the crowd with respect to response rates, reporting on the distribution of the number of tasks submitted per crowd worker, and their performance in terms of speed of acceptance and completion of tasks. It is impossible to accurately measure the time a crowd member specifically devotes to a task, because we cannot know whether they are solely focused on the task. Nevertheless, we can report on the time between creation and acceptance of the task, between acceptance and submission of the task and also the total logged time that a crowd member spends working on a task.

RQ2: The crowd's coverage attainment: We use the crowd as a source of test data in its own right, as well as the ability of the crowd to provide observations from which we can extract motif patterns for automated test techniques. In order to investigate the crowd's value as a source of test data in its own right, we report on the coverage obtained by the crowd as the number of tasks completed increases. We report both the overall number of unique and non-unique activities covered, and also the level of activity coverage per subject, for each of the nine subject apps under test.

Having investigated the demographics and behaviour of the crowd of their ability to generate test data, we move on to consider the relationship between crowd-based testing and automated testing. In particular, we compare crowd testing for Android, with a recently proposed, state-of-the-art technique, SAPIENZ [7], for automated test data generation for Android using search based software testing. We first compare the crowd's and SAPIENZ' coverage attainment, in terms of unique activities covered, and then investigate the degree to which motif patterns, extracted from the crowd (using our motif-extraction algorithm) can improve the coverage performance of SAPIENZ.

RQ3: The comparative activity coverage achieved by the crowd and by SAPIENZ: In order to answer this question, we report on the number of unique activities covered by the crowd and by SAPIENZ, and their intersection. This allows us to explore the degree to which the two techniques are complementary to one another, and also the degree of overlap between automated testing and crowd-based testing. Should it turn out that the automated technique subsumes the human-based technique, then there would be little point in investigating motif pattern extraction, but if crowd testing can achieve better or different (complementary) coverage, then this suggests that there may be scope to improve automated test data generation with motif pattern extraction from observed crowd-based tests. The current version of POLARIZ records coverage but not faults found; future work will extend it to record detailed fault context.

RQ4: The improvement in SAPIENZ performance when using motif patterns extracted from crowd-based tests: Finally, we investigate the degree to which SAPIENZ' coverage is improved, for each of the nine apps under test, when SAPIENZ is imbued with information extracted from the crowd-based testing in the form of motif patterns.

TABLE I
NINE POPULAR GOOGLE PLAY SUBJECT APPS ('#A' FOR NUMBER OF ACTIVITIES; '#M' FOR NUMBER OF METHODS; 'INSTALLS' IS IN MILLIONS)

Subject	Ver.	Category	Description	#A	#M	Rating	Installs
HP All-in-One Printer Remote	4.1.18	Productivity	Help users scan and print documents with HP printers.	74	13,616	4.1	10-50M
TuneIn Radio	17.1	Music & Audio	Let users listen to radio stations for free.	27	13,474	4.4	100-500M
Trainline	2.5.0	Maps & Navigation	A railway information provider.	41	7,497	4.3	1-5M
Power Security	1.0.18	Tools	Scan and kill viruses, malware and spyware.	38	5,802	4.4	5-10M
Google Translate	5.6.0	Tools	Translate between 103 languages.	17	4,765	4.4	100-500M
Brightest Flashlight	1.35	Productivity	A multi-functional flashlight app.	27	6,087	4.3	5-10M
Duolingo	3.39.1	Education	Learn multiple languages fast, fun and free.	29	9,949	4.7	50-100M
Clean My Android	1.1.9	Productivity	A light phone cleaner and app manager.	16	1,804	4.7	1-5M
Citymapper	6.15	Maps & Navigation	A journey planner and route finder.	32	25,998	4.5	1-5M

A. Subject Applications

We perform the empirical evaluation on 9 randomly-selected real-world Google Play apps from top 500 most popular free/in-app-purchase apps as listed in the Google Play app store on December 20, 2016. We chose 9 subjects from a list of all apps, filtered based on their availability for our hardware resources (9 Nexus-7 tablets in the POLARIZ device infrastructure), and constraints imposed by a desire to use the subjects in experiments on the crowd-based testing.

That is, when we perform the random selection, we first exclude gaming apps that are not based on standard Android native UI components. Also, to protect the crowd testers' privacy, we also exclude apps that request user account information after launching. The crowd was also notified that they should not disclose any personal information during the testing process.

The 9 apps that were selected randomly after this filtering process, are closed-sourced and cover multiple app categories. Each app has at least 1 million user installs (according to Google Play). Detailed subject information including version numbers, sizes, ratings and the number of installs are presented in Table I.

B. Experimental Settings

For each subject, we assign the same app running environment, i.e., the same software and hardware configurations. These configurations mimic general real-world end-user testing scenarios, e.g., with real devices that have Google service framework and WIFI network connection, but without providing app-specific contexts. For example, the 'HP All-in-One Printer Remote' app may require an HP printer for testing some of its functionalities. In our experiments, we do not provide such app-specific equipment for the generalisation purpose.

We also need to recruit crowd workers and manage payments by using a third-party intermediary. We report on our approach to tackling these issues in the remainder of the section in order to support replication and to give the context to the results we present for crowd-based testing.

Crowd recruitment: We use Amazon Mechanical Turk[2] (AMT) for recruiting non-professional crowd workers from the general public. AMT is currently one of the most popular crowdsourcing platforms for micro tasks with general crowd workers. We recruit AMT workers to perform remote mobile testing tasks on our POLARIZ platform by posting human intelligence tasks (HITs) on AMT. Anyone, from any country, who is eligible to work on AMT is allowed to work on our HIT assignments. We only disclose the task information and our POLARIZ web service URL via the AMT HIT for controlling the worker sources (i.e., only AMT workers are expected) because this may interfere the recruitment speed and POLARIZ' visitor statistics.

To motivate the crowd, we provide 1.5 USD payment for each approved submission, as the extrinsic incentive to the crowd workers. We expect each worker will spend 10 minutes or less on one HIT assignment. The payment rate is higher than current UK national minimum wage (7.2 GBP/hour) and also the US standard (7.25 USD/hour). Intrinsic incentives include the opportunity to experience manual mobile testing, and maybe, also to test the remote apps for fun (we investigate the task interest level in the results section).

Task design and quality control: A clear task description is considered to be one of the most important factors for successful software crowdsourcing tasks [14]. This motivates our careful design of our HIT. The general workflow of our designed HIT task is as follows:

1) The crowd worker views the task assignment description on AMT and can choose to accept or decline the task.
2) The worker follows the task instruction and works on our POLARIZ platform via any devices with a browser and performs manual testing on one arbitrarily selected app.
3) Upon finishing the testing task, the worker copies the POLARIZ generated app execution log as the proof of task completion and goes back to the AMT HIT.
4) The worker submits the automatically generated log and answers a questionnaire which contains 6 brief questions.
5) The worker waits for requester's review and gets paid via AMT, assuming their submission meets our sanity check for appropriate engagement.

[2]https://www.mturk.com

In the task description, for comprehensive testing, we instruct the workers that the goal is to explore and trigger as many functionalities of the subject app as possible. A few detailed steps for accessing our POLARIZ platform are illustrated using snapshots. In the questionnaire, we ask 6 short questions to collect their feedback on the interest level of the task, and background information regarding the workers, including their daily mobile usage duration, software testing experience, country, gender and education level. No personal information that may reveal the worker's personal identity is collected.

For quality control, we give three criteria to the workers, which form our 'sanity check' for task approval and consequent payment: First, the worker has tried to explore and trigger multiple app functions (preferably as many as possible). Second, the worker has tested the app for at least 3 minutes. Third, the submitted app execution log contains at least 300 lines. Normally these criteria can be easily satisfied by testing the app for a few minutes. We review each submission by checking above three criteria in a semi-automated manner. We measure these three criteria based on the submitted logs (for Criteria 1, we use at least two activities as the lower bound). As a further sanity check, we also manually inspect the submissions periodically. If a submission is rejected, we do not repost that assignment.

We posted 1,350 assignments from December 22, 2016 to January 2, 2017, in a continuous manner, in order to leave time to perform daily reviews. These 1,350 assignments were split into 150 HITs, each containing 9 assignments. All HITs and their assignments are the same. Each HIT may contain one or more task assignments. Each worker can work on multiple HITs, but can only work on one assignment in one HIT. Our quality control filter removed 20.4% of these HITs to leave 1,075 for subsequent testing and motif extraction.

POLARIZ deployment: We deploy POLARIZ as a publicly accessible web service, at a server located in the UK, plus a Linode cloud server as a proxy to speed up global visits. The mobile device infrastructure is hosted at the author's lab and connected to the front-end server. Accessing the remote device does not require authentication but the mobile devices are monitored and manipulated under POLARIZ' permission control component, where changes to environment settings are prohibited.

The 9 subjects are pre-installed on 9 Nexus-7 tablets; one per device. User interactions are logged with timestamp information which can be mapped to the submitted logs. All subjects are reset to their initial states every half an hour. This is to avoid the case that one worker drives the app into a state from which the subsequent workers cannot recover. Of course, we could have chosen reset app state per worker, but we found that multiple workers can collectively work on one AUT, by setting the reset duration to 30 minutes.

Performance metrics: We measure coverage attained, since this is a fundamental metric for testing [15]–[17]. In terms of granularity of coverage, we measure app activity coverage; an approach to coverage measurement that has been adopted in previous studies on automated mobile testing [7], [18].

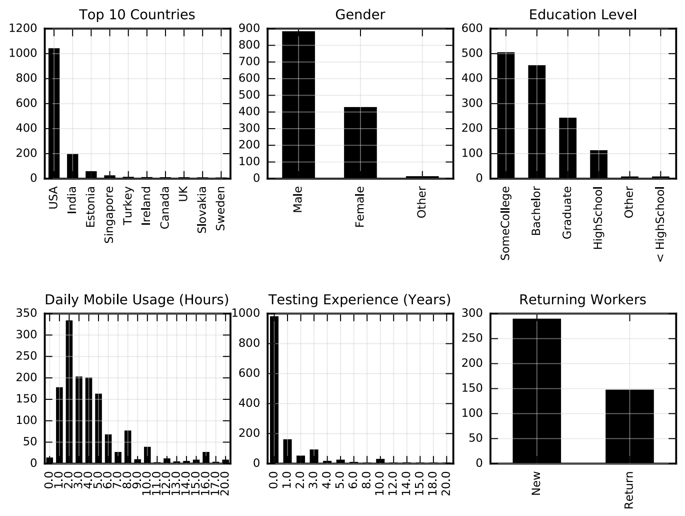

Fig. 3. Crowd worker demographic information based on 1,350 submitted assignments

This metric thus gives us a baseline against which to assess and compare the ability of tests to 'explore' the AUT.

Motif extraction: We learn generalised event patterns because high-level events learned from only a single subject has already been proved useful in the literature [19]. In our experiment, we perform a leave-one-out evaluation on the extracted crowd motifs. That is, when evaluating a subject, we will use only the motifs extracted from the remaining 8 subjects' event traces. For each subject, we apply the POLARIZ motif extraction algorithm to learn three motif patterns.

Improving SAPIENZ: To examine whether the learned generalised motifs are helpful in improving app activity coverage, we run SAPIENZ without any motif information and compare results to these obtained from running SAPIENZ with the learned crowd motifs. On each subject, we run SAPIENZ for 60 minutes wall-clock time. We set the delay between each two events to 200 ms so that, given the same amount of wall-clock time, roughly the same number of events will be used. This setting aims for a fair comparison between the two SAPIENZ versions with and without motif patterns. For SAPIENZ parameters, we use the default settings, as reported by the authors of the SAPIENZ paper [7]. In all experiments, the parameters were not tuned, to avoid any implicit experimental biases that might otherwise arise.

We run all experiments on the same MacBook Pro with 2.3GHz Intel Core i7 CPU and 16G RAM. The mobile side for app execution is a Nexus-7 real device.

IV. RESULTS

The results show that POLARIZ successfully assisted the crowd workers to complete all 1,350 AMT tasks, from December 22, 2016 to January 2, 2017. We also find evidence to support the claim that there is complementarity between the crowd-based tests and search based tests found by SAPIENZ. We further report evidence to support the claim that motif patterns, extracted using or algorithm, can improve the attainment of activity coverage by SAPIENZ.

A. RQ1: Demographics and Behaviour

RQ1 is decomposed into three sub-questions concerning demographics, interest level and crowd behaviour, each of which we report on below.

RQ1.1: The demographic diversity of the crowd: According to visitor tracking data from Google Analytics, from December 22nd. 2016 to January 2nd. 2017, there were 1,931 sessions of visits to our remote crowd testing service. Of these sessions, 56.9% come from new visitors and 43.1% from returning visitors. The records show the traffic comes from at least 9 countries, with most coming from the USA (60.90%) and India (24.91%). Note that there are other countries with large populations (such as China) whose workers are ineligible to work on AMT, so there are no visits from these countries. The average response times (from the 14 global sites accessed by POLARIZ during the course of our empirical study) range from 0.465 ms (London) to 295.921 ms (Sydney). This result indicates a reasonably good connectivity of our distributed infrastructure for performing remote testing over a wide range of geographical locations.

During the 12 days of experimentation time, our 1,350 posted HIT assignments were all finished by the crowd workers. Of all submitted solutions, 1,075 (79.6%) were approved, according to the criteria for quality control discussed in Section III-B. We received 1,350 submissions from 434 distinct workers. Results from our questionnaire show that these workers come from 24 countries, while 99% submissions are from the top 10 most frequently submitting countries (as listed in Figure 3). Note that the number of countries is inconsistent with the traffic we observed according to Google Analytics; the questionnaire reveals a far wider country participation than that would be suggested by the Google Analytics data. Our interpretation is that a small number of AMT workers may use proxies to visit the AMT (to overcome the fact that the service is disabled in their countries).

Figure 3 presents worker demographic information based on the 1,350 responses submitted by the crowd. This self-assessment is broadly consistent with the analytics data reported by Google; most workers come from USA (76.7%) and India (14.1%). However, using the self-assessment questionnaire, we were able to obtain further demographic information: more male workers (64.4%) submitted than female workers (31.4%). Regarding the educational level, 88.3% workers at least attended some college education (including undergraduate students). This generally high educational level is consistent with previous studies [20], [21], although our results show that there are more workers with some college education than those holding a Bachelor's degree.

Since our remote testing tasks require basic skills for interacting with mobile apps, we expected the crowd to have regular (daily) mobile usage. Our questionnaire results on 'Daily Mobile Usage' suggest that only 0.9% of the respondents spend less than 1 hour per day on mobile usage, indicating that our expectation is reasonable.

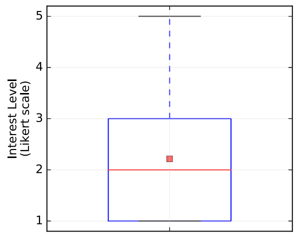

Fig. 4. Worker feedback on self-assessed interest level of the task (1 = Very Interesting; 5 = Very Boring)

We also recruit the crowd from the general public rather than software testing experts. As the distribution presented in Figure 3 indicates, 72.3% respondents have less than one year testing experience, and the remaining 27.7% have at least one year's experience in software testing. Given that we recruit from the general public, a proportion of over a quarter having testing experience was a surprise to us (since testers do not occupy 1/4 of the world's population).

Our understanding is that their experience may come from working on testing tasks posted on AMT or other crowd testing platforms such as uTest or they may have professional career experience in software testing. Furthermore, those with testing experience may favour our HIT, while those without such experience may have self-selected out. It is interesting to note that an open call with no pre-requisites for test experience still ends up recruiting a crowd with higher-than-average testing expertise.

Finally, we observe that 66.4% of the crowd workers recruited are 'new'. That is, they only completed one task, while the remaining workers completed at least two tasks. This high rate of returning workers may be correlated with the interest level of our task: a topic to which we now turn.

RQ1.2: The crowd's interest level: Figure 4 shows the feedback from the crowd on the interest levels of our task. The boxplot suggests a mean rating of 2.3 (between 2-'Interesting' and 3-'Normal'; lower values to note higher interest levels), and a median rating of 2. This relatively high rating of interest level may explain the high rate of returning workers revealed in the results of Figure 3. A detailed distribution of the number of submitted tasks by each worker is given in Figure 5. The distribution shows that, although there is a high rate of returning workers, the total submissions are not dominated by a small number of 'super workers', thereby giving cause for optimism regarding the crowd's diversity.

RQ1.3: The crowd's response rates: We investigated crowd performance along two dimensions, the speed of task performance and the thoroughness of crowdsourced manual testing in terms of activity coverage. The speed data is extracted from AMT task records and also the app execution logs submitted by the crowd. The app activity coverage data is calculated based on the submitted logs, which are produced by Android LOGCAT. The log information contains detailed activity launch, warning and error information.

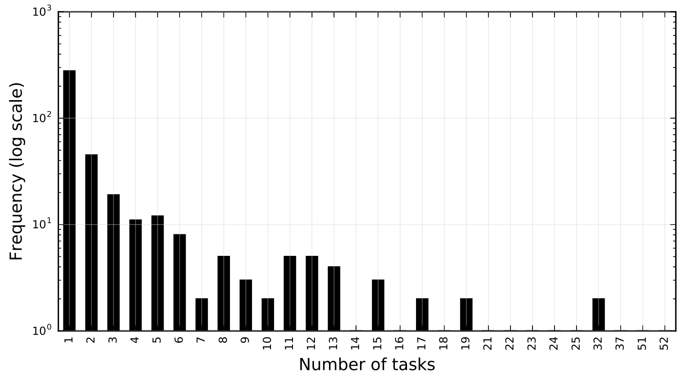

Fig. 5. Task distribution per worker

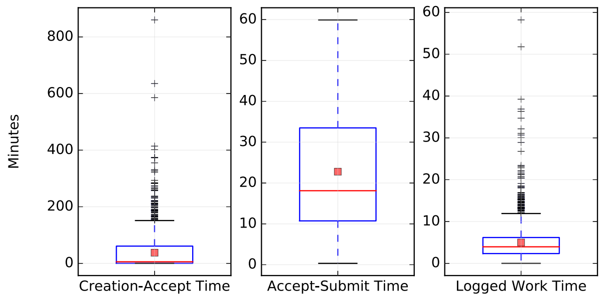

Fig. 6. Task acceptance and completion times

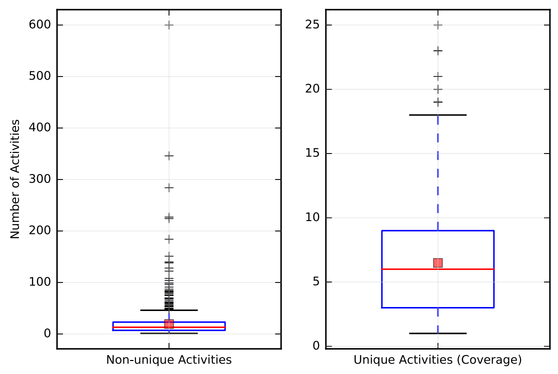

Fig. 7. Number of covered activities per task

Many testing scenarios may be sensitive to test speed. Figure 6 presents three boxplots on the crowdsourced mobile testing enabled by POLARIZ. The 'Create-Accept' time is the elapsed time from posting a task on AMT to a worker consenting/accepting to work on the task. In the first boxplot, the time for the 75th percentile is 61.0 minutes and 73.3% of the posted tasks were accepted within one hour. The 'Accept-Submit' time reports the time from task acceptance to submission of a solution by the crowd worker. The second boxplot reveals that all 1,350 posted tasks finished within one hour, with a median value of 18.1 minutes.

Note that this time cost may not reflect the actual working time, because the worker may simply accept the task, and work on something else first. Thus we regard the data presented in the second boxplot as an upper bound on the working time. To further examine the lower bound, we check the crowd's working time based on the logs submitted. The logged time may not reflect the time required to become familiar with our POLARIZ platform, thus we regard it as a lower bound. As shown in the third boxplot in Figure 6, the interquartile range (25th to 75th percentiles) area shows a range of 2.3 to 6.2 minutes, which falls into our expectation on the working time, i.e., within 10 minutes.

RQ2: The crowd's level of coverage attainment: The crowd's performance in terms of test coverage attained is shown in Figures 7 to 9. First we examine the overall coverage and then consider the detailed coverage results for each of the subjects.

Figure 7 shows boxplots that depict the number of covered unique and non-unique activities per task. For non-unique activities, the number of triggered activities is 7 to 23 for the interquartile range, while the number for unique activities is 3 to 9. Considering the real-world complexity of the subjects, and the fact that our testing tasks are designed to be lightweight/micro tasks, this coverage performance is reasonable and is within our expectation.

Figure 8 shows the crowd's cumulative coverage over all 9 subjects. The horizontal axis represents tasks in chronological submission order, while the vertical axis reports activity coverage. In total, 21,440 non-unique activities were manipulated by the crowd, which covered 182 (out of 301 total) unique activities over all 9 subjects; 60.5% unique activity coverage.

Figure 9 reports the coverage achieved on each of the 9 subjects. Each subject is randomly assigned, so the x-axis for the number of tasks may vary slightly between subjects, but each subject corresponds to at least 100 tasks. In all 9 cases, the cumulative coverage grows rapidly for the first 10 tasks and subsequently 'plateaus out'. In a few cases (e.g., 'TheTrainline'), the coverage was still able to grow after more than 100 tasks have been considered.

The highest coverage is achieved on the 'CleanMyAndroid' subject (87.5%). While the lowest coverage is on the 'All-in-One Printer Remote' subject (21.6%), which is the only subject with a coverage below 60%. This low coverage is caused by the app-specific contexts which require external hardware to be present, such as connecting to a HP printer. In our experiments, such external hardware was unavailable.

RQ3: The comparative activity coverage achieved by the crowd and by SAPIENZ: We map the SAPIENZ coverage for each subject to the coverage achieved by the crowd, as the Venn diagrams illustrate in Figure 10. From the 9 Venn diagrams we can see that the crowd covered more app activities than the fully automated SAPIENZ approach in 8 of the 9 cases. As expected, the superior to main knowledge of the crowd and the high-level understanding of the purpose of the apps under test gives them an advantage in covering activities, compare to (cheaper) fully automated techniques.

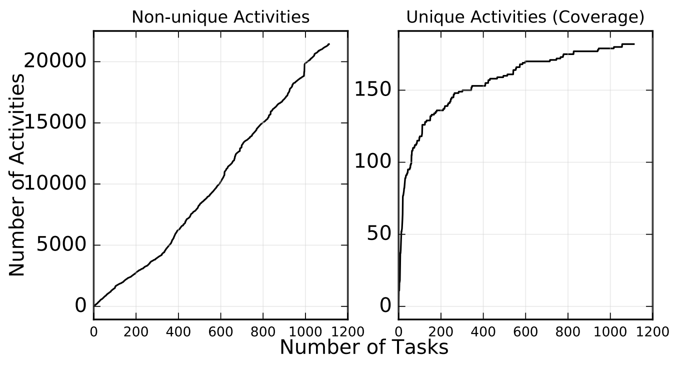

Fig. 8. Overall cumulative crowd test coverage

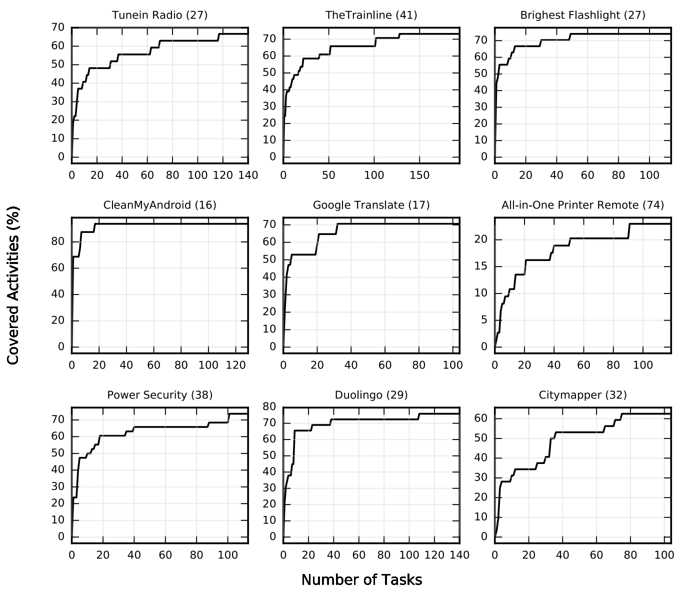

Fig. 9. Crowd test coverage by subject, with total number of activities (in parentheses).

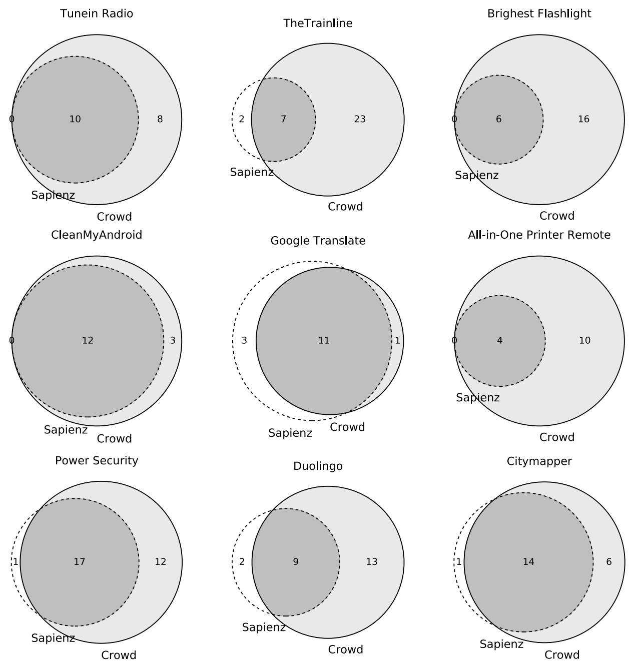

Fig. 10. Activities by Sapienz and the crowd

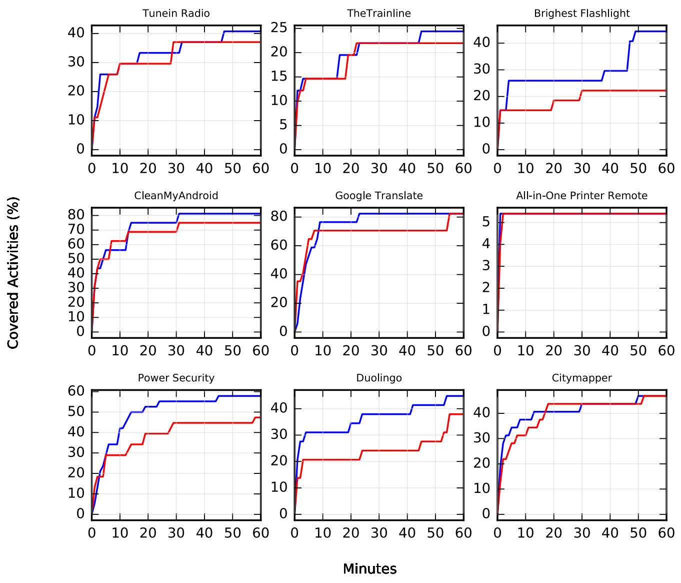

Fig. 11. Improvement in coverage. Lower (red/lighter gray) lines denote Sapienz without motif patterns.

There is only one case (Google Translate), for which SAPIENZ triggered more activities than the crowd. However, Figure 10 also reveals that the two approaches complement one another in 5 out of the 9 cases, including the Google Translate case.

RQ4: The improvement in SAPIENZ performance when using motif patterns extracted from crowd-based tests: We analyse the effectiveness of the crowd motifs learned by our POLARIZ motif extraction algorithm. On all 9 subjects, we confirmed all the learned motif events were, indeed, reused by Sapienz. In Figure 11, we draw the coverage achieved by both SAPIENZ with and without the learned motif events, where the blue (darker grayscale, when viewed in black and white) lines indicate the performance of SAPIENZ with motifs, and the red (lighter grayscale) lines denote results for SAPIENZ without a motif. As suggested by the line charts of cumulative coverage on each of the subjects, the learned motifs were able to enhance SAPIENZ in achieving higher test coverage in 6 out of 9 cases.

In the remaining 3 cases, the integrated motif patterns led to neither improvement nor disimprovement in terms of app activity coverage. However, in the best case (Brightest flashlight), activity coverage improved by 100%, rising from 6 to 12 unique activities covered out of 27 total possible unique activities.

The parameters used in the experiments have not been tuned so our results represent fair lower bounds on the improvement that could be expected to accrue; parameter tuning and more ‘targeted’ learning might improve the results we report here.

B. Threats to Validity

The primary threat to validity of our empirical studies is the threat to external validity. Our subject dataset excluded two types of mobile apps, i.e., games (with non-standard Android UI components) and those having an initial login activity (for protecting the crowd's privacy). Our results may therefore fail to generalise to these kinds of apps.

To partly mitigate the generalisation issue, we randomly chose apps, and we note that they did fall into multiple app categories from widely-installed real-world apps, each of which has at least 1 million installs. We also cannot be sure that the improved coverage observed for SAPIENZ would necessarily be observed for other automated testing approaches. We also found that our crowd contains a surprisingly high level of testing expertise for the 'general public', a characteristic that may also fail to generalise to other scenarios

To minimise internal threats to validity, we tested both the components of POLARIZ and the scripts for data collection and analysis. One threat to internal validity that we cannot avoid is related to the permission control component of POLARIZ platform: To guarantee that the app testing contexts (e.g., WIFI connection) will not be changed by the crowd workers, the permission component disallows any call to external activities that are not part of the app under test.

It is possible that, for certain subjects, such calls to external activities are a precondition to trigger some of their own activities. Although the same restriction applies to both techniques studied, we cannot discount the possibility that such security-sensitive blocking might have disproportionately affected one or other of our two treatments.

V. Related Work

Our work is most closely related to previous work on extraction of useful patterns for app testing, crowdsourcing and automated test generation, the three areas it combines.

Pattern Extraction: Linares-Vásquez et al. introduced MonkeyLab [5]. Like MonkeyLab, POLARIZ extracts patterns from app usage data. However, unlike MonkeyLab, POLARIZ exploits a crowdsourced model which is context free (whereas MokeyLab is concernewd with context in its model building). Furthermore, while MonekyLab focuses on extraction of value for a single app under test and is agnostic about its downstream use, POLARIZ introduces a novel crowdsourcing platform and extraction algorithm that targets common patterns extracted from (and for) multiple apps, for subsequent exploitation by the specific downstream application of automated test data generation.

Crowdsourced Testing: Crowdsourcing is increasingly popular in software engineering research [22]–[27]. Previous work on crowdsourced software testing has formulated the test design problem as one to be outsourced to the crowd. For example, Dolstra et al. [28] and Vliegendhart et al. [29] demonstrated the usefulness of using Amazon Mechanical Turk workforce to perform continuous GUI testing.

Schneider and Cheung [30] proposed to employ on-demand crowd users for usability testing. Chen and Kim [31] proposed a Puzzle-based Automatic Testing (PAT) technique that transforms object mutation problems into puzzles for the crowd to solve. Pastore et al. [32] used crowdsourcing to tackle the oracle problem [9]. In this previous work, crowdsourcing is used as an independent source of test data, whereas our approach uses the crowd to help guide automated testing.

Automated Test Generation: There exist several mature semi-automated testing frameworks such as Appium [33] and Robotium [34] that are widely used in industry, but these frameworks automate capture and replay, but not test case design. By contrast, fully-automated mobile test generation research prototypes have rarely proved able to outperform random testing [3], [35]. For example, Dynodroid [36] uses a biased random strategy, while SwiftHand [37] and ORBIT [38] and PUMA [39] used model-based approaches. Other approaches such as ACTEve [40] and TrimDroid [41] are based on program analysis. Nevertheless, the coverage achieved by the state-of-practice tool Android Monkey has tended to achieve higher coverage than all of these research prototypes, according to recent empirical results [3]. EvoDroid [42] was the first search-based software testing system for Android reported in the literature. In this work we chose to use SAPIENZ [7], partly because it is publicly available (unlike EvoDroid), but primarily because it has been recently demonstrated to significantly outperform both the state-of-the-art automated testing (Dynodroid [36]) and the state-of-practice (Android Monkey). We thus used SAPIENZ in order to ensure that our approach can further improve on the current best-obtainable results for automated Android testing. This allows us to be sure that our approach advances the current state-of-the-art in automated testing by hybridising with mining from crowdsourced usage patterns.

Compared to this previous work POLARIZ is the first to combine automated (search-based) testing and crowdsourcing and also the first to leverage cross-app usage patterns for improved mobile testing. Our results demonstrate that this combination complements and extends the state-of-the-art in search based testing.

VI. Summary

We introduced the POLARIZ approach to crowd-based testing, which leverages a non-professional crowd to provide test cases from which we extract motif patterns to help guide the SAPIENZ automated testing technique. Our evaluation on 9 popular Google Play apps showed that POLARIZ was able to harness 434 crowd workers from 24 countries to perform 1,350 testing assignments. The automatically-learned motif patterns improved SAPIENZ' activity coverage of 6 out of 9 subjects, leaving it no worse on the remaining 3. We also found that SAPIENZ and crowd-based approaches complemented one another in 5 out of 9 subject apps, further motivating approaches, such as ours, that seek to combine them.

References

[1] C. Cadar and K. Sen, "Symbolic execution for software testing: Three decades later," *Communications of the ACM*, vol. 56, no. 2, pp. 82–90, February 2013.

[2] M. Harman, Y. Jia, and Y. Zhang, "Achievements, open problems and challenges for search based software testing," in *Proc. of ICST'15*, 2015, pp. 1–12.

[3] S. R. Choudhary, A. Gorla, and A. Orso, "Automated test input generation for Android: Are we there yet?" in *Proc. of ASE'15*, 2015, pp. 429–440.

[4] M. Bozkurt and M. Harman, "Automatically generating realistic test input from web services," in *Proc. of SOSE'11*, 2011, pp. 13–24.

[5] M. Linares-Vásquez, M. White, C. Bernal-Cárdenas, K. Moran, and D. Poshyvanyk, "Mining Android app usages for generating actionable GUI-based execution scenarios," in *Proc. of MSR'15*, 2015, pp. 111–122.

[6] K. Moran, M. Linares-Vásquez, C. Bernal-Cárdenas, C. Vendome, and D. Poshyvanyk, "Automatically discovering, reporting and reproducing Android application crashes," in *Proc. of ICST'16*, 2016, pp. 33–44.

[7] K. Mao, M. Harman, and Y. Jia, "Sapienz: Multi-objective automated testing for Android applications," in *Proc. of ISSTA'16*, 2016, pp. 94–105.

[8] L. Gomez, I. Neamtiu, T. Azim, and T. Millstein, "RERAN: Timing- and touch-sensitive record and replay for android," in *Proc. of ICSE'13*, 2013, pp. 72–81.

[9] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE Transactions on Software Engineering*, vol. 41, no. 5, pp. 507–525, 2015.

[10] P. D'haeseleer, "What are dna sequence motifs?" *Nature biotechnology*, vol. 24, no. 4, pp. 423–425, 2006.

[11] M. K. Das and H.-K. Dai, "A survey of dna motif finding algorithms," *BMC bioinformatics*, vol. 8, no. 7, p. 1, 2007.

[12] H. Huo, Z. Zhao, V. Stojkovic, and L. Liu, "Optimizing genetic algorithm for motif discovery," *Mathematical and Computer Modelling*, vol. 52, no. 11, pp. 2011–2020, 2010.

[13] M. Kaya, "Mogamod: Multi-objective genetic algorithm for motif discovery," *Expert Systems with Applications*, vol. 36, no. 2, pp. 1039–1047, 2009.

[14] B. Fitzgerald and K.-J. Stol, "The Dos and Don'ts of Crowdsourcing Software Development," in *SOFSEM 2015: Theory and Practice of Computer Science*, ser. Lecture Notes in Computer Science, 2015, vol. 8939, pp. 58–64.

[15] H. Zhu, P. A. Hall, and J. H. May, "Software unit test coverage and adequacy," *ACM Computing Surveys*, vol. 29, no. 4, pp. 366–427, 1997.

[16] A. M. Memon, M. L. Soffa, and M. E. Pollack, "Coverage criteria for gui testing," *ACM SIGSOFT Software Engineering Notes*, vol. 26, no. 5, pp. 256–267, 2001.

[17] G. J. Myers, C. Sandler, and T. Badgett, *The art of software testing*. John Wiley & Sons, 2011.

[18] L. Clapp, O. Bastani, S. Anand, and A. Aiken, "Minimizing gui event traces," in *Proc. of FSE'16*, 2016, pp. 422–434.

[19] M. Ermuth and M. Pradel, "Monkey see, monkey do: Effective generation of gui tests with inferred macro events," in *Proc. of ISSTA'16*, 2016, pp. 82–93.

[20] J. Ross, A. Zaldivar, L. Irani, and B. Tomlinson, "Who are the Turkers? worker demographics in Amazon mechanical turk," Department of Informatics, University of California, Irvine, USA, Tech. Rep., 2009.

[21] J. Ross, L. Irani, M. Silberman, A. Zaldivar, and B. Tomlinson, "Who are the crowdworkers? shifting demographics in mechanical turk," in *Proc. of CHI'10*, 2010, pp. 2863–2872.

[22] K. Mao, L. Capra, M. Harman, and Y. Jia, "A survey of the use of crowdsourcing in software engineering," *Journal of Systems and Software*, vol. 126, pp. 57 – 84, 2017.

[23] F. Chen and S. Kim, "Crowd debugging," in *Proc. of FSE'15*, 2015, pp. 320–332.

[24] J. Wang, S. Wang, Q. Cui, and Q. Wang, "Local-based active classification of test report to assist crowdsourced testing," in *Proc. of ASE'16*, 2016, pp. 190–201.

[25] L. Ponzanelli, A. Bacchelli, and M. Lanza, "Leveraging crowd knowledge for software comprehension and development," in *Proc. of CSMR'13*, 2013, pp. 57–66.

[26] K. Mao, Y. Yang, M. Li, and M. Harman, "Pricing Crowdsourcing Based Software Development Tasks," in *Proc. of ICSE'13 (NIER Track)*, 2013, pp. 1205–1208.

[27] K. Mao, Y. Yang, Q. Wang, Y. Jia, and M. Harman, "Developer recommendation for crowdsourced software development tasks," in *Proc. of SOSE'15*, 2015, pp. 347–356.

[28] E. Dolstra, R. Vliegendhart, and J. Pouwelse, "Crowdsourcing GUI tests," in *Proc. of ISSTA'13*, March 2013, pp. 332–341.

[29] R. Vliegendhart, E. Dolstra, and J. Pouwelse, "Crowdsourced user interface testing for multimedia applications," in *Proc. of CrowdMM'12*, 2012, pp. 21–22.

[30] C. Schneider and T. Cheung, "The power of the crowd: Performing usability testing using an on-demand workforce," in *Information Systems Development*. Springer, 2013, pp. 551–560.

[31] N. Chen and S. Kim, "Puzzle-based automatic testing: Bringing humans into the loop by solving puzzles," in *Proc. of ASE'12*, 2012, pp. 140–149.

[32] F. Pastore, L. Mariani, and G. Fraser, "CrowdOracles: Can the crowd solve the oracle problem?" in *Proc. of ISSTA'13*, March 2013, pp. 342–351.

[33] "Appium: Automation for iOS and Android apps," http://appium.io.

[34] "Robotium: User scenario testing for Android," https://github.com/RobotiumTech/robotium.

[35] X. Zeng, D. Li, W. Zheng, F. Xia, Y. Deng, W. Lam, W. Yang, and T. Xie, "Automated test input generation for android: Are we really there yet in an industrial case?" in *Proc. of FSE'16*, 2016, pp. 987–992.

[36] A. Machiry, R. Tahiliani, and M. Naik, "Dynodroid: An input generation system for Android apps," in *Proc. of ESEC/FSE'13*, 2013, pp. 224–234.

[37] W. Choi, G. Necula, and K. Sen, "Guided GUI testing of Android apps with minimal restart and approximate learning," in *Proc. of OOPSLA'13*, 2013, pp. 623–640.

[38] W. Yang, M. R. Prasad, and T. Xie, "A grey-box approach for automated GUI-model generation of mobile applications," in *Proc. of FASE'13*, 2013, pp. 250–265.

[39] 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 *Proc. of MobiSys'14*, 2014, pp. 204–217.

[40] S. Anand, M. Naik, M. J. Harrold, and H. Yang, "Automated concolic testing of smartphone apps," in *Proc. of FSE'12*, 2012, pp. 59:1–59:11.

[41] N. Mirzaei, J. Garcia, H. Bagheri, A. Sadeghi, and S. Malek, "Reducing combinatorics in GUI testing of Android applications," in *Proc. of ICSE'16*, 2016, pp. 559–570.

[42] R. Mahmood, N. Mirzaei, and S. Malek, "EvoDroid: Segmented evolutionary testing of Android apps," in *Proc. of FSE'14*, 2014, pp. 599–609.