Search-Based Test Generation for Android Apps

Iván Arcuschin Moreno
Universidad de Buenos Aires, Argentina
iarcuschin@dc.uba.ar

Abstract

Despite their growing popularity, apps tend to contain defects which can ultimately manifest as failures (or *crashes*) to end-users. Different automated tools for testing Android apps have been proposed in order to improve software quality. Although Genetic Algorithms and Evolutionary Algorithms (EA) have been promising in recent years, in light of recent results, it seems they are not yet fully tailored to the problem of Android test generation. Thus, this thesis aims to design and evaluate algorithms for alleviating the burden of testing Android apps. In particular, I plan to investigate which is the best search-based algorithm for this particular problem. As the thesis advances, I expect to develop a fully open-source test case generator for Android applications that will serve as a framework for comparing different algorithms. These algorithms will be compared using statistical analysis on both open-source (i.e., from F-Droid) and commercial applications (i.e., from Google Play Store).

Keywords

Android testing, search-based, evolutionary algorithms

ACM Reference Format:
Iván Arcuschin Moreno. 2020. Search-Based Test Generation for Android Apps. In *42nd International Conference on Software Engineering Companion (ICSE '20 Companion), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3377812.3381389

1 Research Problem

As software continues increasing its importance in our daily lives, the use of mobile devices such as smartphones and tablets increases as well. It is estimated that mobile technologies are now used by two-thirds of the global population [4]. In this context, smartphones have become the dominant platform for mobile time consumption, in terms of total minutes across every market. About 80% of all mobile time [3] is spent in application consumption (commonly known as "apps"). As of November 2019, there are over 2.8 million applications available on Google's Play App Store [7].

ICSE '20 Companion, May 23–29, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7122-3/20/05…$15.00
https://doi.org/10.1145/3377812.3381389

Despite their growing popularity, apps tend to contain defects which can ultimately manifest as failures (or *crashes*) to end-users. Similarly to other software, testing mobile apps allows developers to ensure a minimum quality threshold for the applications they write. This process typically involves manually writing test cases. Testing intends to assure that new features behave as expected and that changes to the source code do not break previous existing functionality. However, testing is a very time consuming and error-prone task, and hence expensive [21].

Different automated tools for testing Android apps have been proposed in order to improve software quality [15, 32]. Monkey [8], regarded as the state-of-practice, is a popular random-based testing tool for Android applications. It is provided with the Android SDK, but it only reports uncaught runtime exceptions during a random exploration. Furthermore, Monkey does not allow users to replay sequences of events (i.e., test cases), which is critical for understanding the source of the regression fault. Dynodroid [23] is a tool designed to overcome the limitations of Monkey that extends pure random testing with two feedback directed biases: *Biased Random*, which uses context adjusted weights for each event, and *Frequency*, which has a bias towards least recently used events.

Currently, there are two tools regarded as state-of-the-art in the literature: Sapienz [24] and Stoat [30]. On one hand, Sapienz [24] is a test generator that has applied search-based testing to Android apps. Its main distinctive feature is the usage of a multi-objective evolutionary algorithm. Mao et al. [24] have shown that it can outperform both Monkey and Dynodroid. This success led to its recent production deployment at Facebook [2] where it is used to automatically generate test cases [9]. On the other hand, Stoat [30] is a test generator that has applied model-based evolutionary testing. It uses dynamic exploration to build and evolve a probabilistic UI state-transition model from which to gather test cases. Ting Su et al. [30] have shown that it can outperform Sapienz.

These works inspire the main question of my thesis: **"Which is the best search-based algorithm to use for Android test generation?"**. Although fairly simple to formulate, it yields several secondary questions that I will need to address before being able to answer the former.

Should we use population-based evolutionary algorithms (e.g., NSGA-II [16]), as it is traditional in the test generation literature, or a completely different search-based technique from the broader field of meta-heuristics (e.g., Simulated Annealing [31])? In particular, as I will explain in Section 5, there are strong reasons to believe that using population-based evolutionary algorithms might not be the best fit for Android test generation. What is more, most of the evolutionary algorithms presented in the testing literature are tailored towards

unit test generation (e.g., like the ones used in EvoSuite tool [19]), but in recent years there have been some proposals for system-level test generation (e.g., MIO [10]) which might be worth exploring.

What is the best internal representation of individuals for these algorithms? Different tools have made different choices. For example, Sapienz uses *atomic* actions (e.g., pressing down a key, taping the screen at a given certain coordinate, etc.) while tools like MATE [17] use *widget* actions (e.g., clicking a *button* on the screen). This choice is very important since using one or the other will yield different search spaces. To the best of my knowledge, there are no studies on the impact of this decision on algorithms' effectiveness.

What is the best output format for an Android *test generation tool?* This is important because reading and saving the generated test cases might prompt developers to write similar ones. As an example, Sapienz outputs a sequence of *atomic* actions that are intended to be used by machines (i.e., for re-executing a failing case), and Stoat only yields bug reports. In both cases the output, although useful, is not readable nor understandable by a human, and as such developers are not able to add these automatically generated "tests" to their source code repositories to build a proper regression test suite.

2 Research Hypothesis

The Sapienz approach presented in Mao et al. [24] distinguishes from previous Android testing tools due to these two features:

i) A multi-objective evolutionary algorithm (NSGA-II [16]) that generates test sequences, simultaneously maximizing statement coverage and fault detection while minimizing test length.
ii) The representation of test cases as sequences of *atomic* and *motif* actions.

An *atomic* action is an event that cannot be further decomposed and *motif* actions are composed "events" that represent a usage pattern on the app. These patterns follow common user behavior, such as filling-in all text fields in the current view and then clicking a button. As such, *motif* actions are based on the User Interface (UI) information available in the current view.

Since these features (i.e., the NSGA-II algorithm combined with *motif* actions) were presented simultaneously, it would be interesting to study the impact of each of them separately. In particular, I am interested in comparing different choices of evolutionary algorithms for Android test generation. This has been already studied for Java unit test generation [14], but it has not been instantiated for Android *system level* test generation. As a system test execution is much more costly than executing unit tests, the effectiveness of a given evolutionary algorithm can be degraded (e.g., because it cannot achieve enough generations to converge in the allotted time). Also, it has been shown that (at least for unit test generation), due to flat fitness landscapes and often simple search problems, Random Search [22] can perform as well as evolutionary algorithms, and sometimes even outperform them [29]. Thus, I would also like to study the choice of Random Search for Android test generation. The initial work I have conducted so far, outlined in Section 5, presents some evidence that the search-based approach used in Sapienz is not statistically better than Random Search. These results are aligned with empirical studies such as the one conducted by Sell et al. [28].

Regarding the internal representation of individuals, my current hypothesis is that using a *widget*-based representation would be more beneficial. Intuitively, if several *atomic* actions can represent the different *"taps"* that can be triggered on a specific button (i.e., at the different coordinates being occupied by such button), they can all be summarized with a single *widget* event (e.g., one simply stating that the button is being clicked). I believe this change in the abstraction level at which the actions take place can have a huge impact on the size and landscape of the search space. In other words, it would facilitate the exploration and help algorithms achieve higher effectiveness.

Finally, concerning the desired output format, my current hypothesis is that tools would benefit from using a more popular framework such as Espresso [5]. This testing framework provides an API that helps developers write concise and readable Android UI tests, and it is officially supported.

3 Expected Contributions

I expect the main theoretical contribution of this thesis to be *a search-based algorithm specifically tailored for* Android *test generation.* This will also involve designing appropriate evolutive operators (i.e., crossover and mutation) so that the search space yields a better, easier to navigate, landscape. Such an algorithm would need to be proven to statistically outperform the current state-of-the-art and state-of-practice techniques. Moreover, the design decisions for this algorithm should be based on careful and methodological experimentation over a broad number of subjects as outlined in Section 4. I also expect to obtain the following artifacts from the successful completion of this thesis.

A new, fully open-source, test case generator for Android *applications.* Initially, this tool will serve as a framework for comparing different algorithms for Android test generation. As the thesis advances, I expect it will become more mature and, hopefully, useful for developers.

An Android Studio *plugin that helps developers to easily run the devised tool and later analyze its output.* As stated in Section 1, Monkey is currently the most popular tool for automatically testing Android applications. I believe Monkey's popularity is partly because it is provided with the Android SDK. In order to smooth the user experience, I will develop a plugin for Android Studio, the official IDE for Android development.

4 Evaluation Procedures

Given that this thesis is focused on designing different algorithms for automatically generating test cases, the choice of proper evaluation techniques to assess such algorithms is

essential to its success. That is, using an inadequate evaluation technique to compare two algorithms can lead to falsely believing that one is better than the other. This would ultimately lead to a misguided research direction.

Taking into consideration that the output of these algorithms are test cases (or test suites), this thesis will use popular metrics already presented in the related literature [12–14, 27, 29, 33], such as: statement coverage, activity coverage (i.e., *screens* coverage), fault-revealing capability and length of the test cases generated. It is important to note that, although there are several studies showing evidence of a relationship between statement coverage and fault detection [18, 25], it is also known that the former does not imply the latter. For this reason, all the studies in this thesis will use both metrics whenever possible.

Since most of these algorithms involve non-deterministic components, it is of paramount importance the use of proper statistical analyses. When comparing different randomized algorithms over a set of subjects, this thesis will follow the same procedures as Campos et al. [14], which in turn are derived from the recommendations outlined by Arcuri et al. [11]. Specifically, this thesis will apply Friedman test [20] with a significance level of $\alpha = 0.05$. The Friedman test is a non-parametric test for multiple-problem analysis and it departs from the traditional tests for significance (e.g., the Wilcoxon test) since it computes the ranking between algorithms over multiple independent problems, i.e., Android applications in our case. A significant $p - value$ indicates that the null hypothesis has to be rejected (i.e., no algorithm in the tournament performs significantly different from the others) in favor of the alternative one (i.e., the performance of algorithms is significantly different from each other). If the null hypothesis is rejected, this thesis will use the post-hoc Conover's test for pairwise multiple comparisons. Such a test is used to detect pairs of algorithms that are significantly different. The $p - values$ obtained with the post-hoc test will be adjusted with the Holm-Bonferroni procedure to correct the statistical significance level ($\alpha = 0.05$) in the case of multiple comparisons.

In the cases where a more detailed comparison between two algorithms in a given subject is needed, this thesis will use the Wilcoxon-Mann-Whitney U-test to determine if there is a statistically significant difference and the Vargha-Delaney A_{12} effect size to measure this difference (if any).

Finally, this thesis will use both open-source (i.e., from F-Droid[1]) and commercial applications (i.e., from Google Play Store[2]) as subjects for the experimentation.

5 Results Achieved so Far

At the beginning of the Ph.D., I performed a large empirical study aimed to deepen into how the main features of Sapienz (namely, the NSGA-II algorithm and the representation of individuals using *motif* actions) impact over effectiveness. In order to achieve this, the study compared the effectiveness of several evolutionary algorithms on 8 experimental subjects. The total execution time was 180 days in a 16 core computer. The evolutionary algorithms considered in the study were taken from the literature [14]: Standard GA, Steady State GA, Monotonic GA, $1 + \lambda, \lambda$ EA, μ, λ EA, and $\mu + \lambda$ EA (as well as the original algorithm implemented in Sapienz: NSGA-II).

To make this comparison fair, all algorithms were implemented on top of Sapienz. Nevertheless, as its authors have stated, the open-source version of Sapienz is regarded as "out-of-date and no longer supported", with the latest activity in the version history recorded in May 2016 [1]. Therefore, I spent some time repairing Sapienz's outdated open-source implementation, fixing some issues such as proper time budget management, handling of timeouts when issuing commands to emulators, recovery from an emulator crash. It is worth noting that this modification of Sapienz is not meant to be a contribution of the Ph.D. It will serve only as a playground for comparing different algorithms.

The experimental results collected have shown that, for the case of Android test generation, the multi-objective NSGA-II algorithm outperforms the other evolutionary algorithms mentioned. However, I also discovered that NSGA-II is not statistically distinguishable from Random Search, which casts doubts about the actual effectiveness of multi-objective evolutionary algorithms for Android test generation. In terms of the impact of *motif* actions, the experimental results provided evidence that both NSGA-II and Random Search performed statistically better when *motif* actions were included. These findings suggest that Sapienz's improvement in effectiveness is due to adding *motif* actions rather than using a particular choice of multi-objective evolutionary algorithms. I would like to continue this research line in the future years by studying the impact that each particular *motif* action has on a broader set of subjects.

Although multi-objective approaches as the one used by Sapienz [24] seemed promising in recent years, in light of our previous results, Genetic Algorithms and Evolutionary Algorithms (EA) in general are not yet fully tailored to the problem of Android test generation. In particular, the results showed that, in our case, the fitness evaluation might take up to 60 seconds for a test case, depending on its length. Overall, this resulted in approximately 30 generations for each EA on average. However, in order to optimize a population towards a given objective, EAs require to evolve as many generations as possible. Therefore, the cost of a fitness evaluation (i.e., executing a system-level Android test on an emulator) directly affects the number of generations the EA can evolve.

Currently, I am working on a tool for automatically generating Espresso test cases. This tool will leverage the test cases generated by tools such as Sapienz and MATE. Although still incipient, I plan on evaluating this tool by generating Espresso test cases for several open-source projects and submitting the generated code via Pull Requests.

[1] https://f-droid.org/en/

[2] https://play.google.com/store

6 Planned Timeline for Completion

Having already completed the first year of my Ph.D., I expect the following timeline for the completion of the remaining tasks in this thesis. It is important to note that some of the following tasks are of a more exploratory nature, i.e. methodically trying different approaches to see what works and what does not.

Second year: Finish the Espresso test generator and its validation experiments. This will involve applying the current prototype on a broader set of subjects and ensuring that it works well for the largest possible number of subjects.

Third year: Explore the different alternatives for the internal representation of individuals. This task will involve using a tool with support for both *atomic* and *widget* actions (e.g., modifying the implementation of SAPIENZ which I have already fixed) and running experiments to understand what is the impact of such decisions on the algorithms' effectiveness. What is more, it would also be interesting to evaluate if the Espresso API mentioned before is suitable to be used for the internal representation (similar to how EVOSUITE [19] uses JUnit [6]).

Fourth year: Explore several novelty search-based approaches presented in recent years for the context of ANDROID test generation. For example, Arcuri [10] has proposed MIO (Many Independent Objectives) which is a technique specially tailored for the context of very large amounts of objectives and limited search budgets (a typical restriction in system/end-to-end testing). In the same paper, MIO was shown to outperform several state-of-the-art multi-objective techniques such as DynaMOSA [26].

Fifth year: Using the knowledge gathered during the thesis, develop a custom ANDROID test generator. As stated in Section 3, I also expect to develop an accompanying ANDROID STUDIO plugin that will ease the use of this tool.

References

[1] 2016. Rhapsod/sapienz: A Prototype of Sapienz (Out-of-date and no longer supported). https://github.com/Rhapsod/sapienz.

[2] 2018. F8 2018: Friction-Free Fault-Finding with Sapienz. https://developers.facebook.com/videos/f8-2018/friction-free-fault-finding-with-sapienz/.

[3] 2018. Global Digital Future in Focus 2018 - Comscore, Inc. https://www.comscore.com/Insights/Presentations-and-Whitepapers/2018/Global-Digital-Future-in-Focus-2018.

[4] 2018. GSMA Intelligence - Research - Global Mobile Trends. https://www.gsmaintelligence.com/research/2018/09/global-mobile-trends/694/.

[5] 2019. Espresso | Android Developers. https://developer.android.com/training/testing/espresso.

[6] 2019. JUnit. https://junit.org/.

[7] 2019. Number of Android applications on the Google Play store | AppBrain. https://www.appbrain.com/stats/number-of-android-apps.

[8] 2019. UI/Application Exerciser Monkey. https://developer.android.com/studio/test/monkey.html.

[9] Nadia Alshahwan, Xinbo Gao, Mark Harman, Yue Jia, Ke Mao, Alexander Mols, Taijin Tei, and Ilya Zorin. 2018. Deploying Search Based Software Engineering with Sapienz at Facebook. In *SSBSE (Lecture Notes in Computer Science)*, Vol. 11036. Springer, 3–45.

[10] Andrea Arcuri. 2019. Many Independent Objective (MIO) Algorithm for Test Suite Generation. *CoRR* abs/1901.01541 (2019).

[11] Andrea Arcuri and Lionel C. Briand. 2014. A Hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. *Softw. Test., Verif. Reliab.* 24, 3 (2014), 219–250.

[12] Andrea Arcuri and Gordon Fraser. 2011. On Parameter Tuning in Search Based Software Engineering. In *SSBSE (Lecture Notes in Computer Science)*, Vol. 6956. Springer, 33–47.

[13] Andrea Arcuri and Gordon Fraser. 2013. Parameter tuning or default values? An empirical investigation in search-based software engineering. *Empirical Software Engineering* 18, 3 (2013), 594–623.

[14] José Campos, Yan Ge, Gordon Fraser, Marcelo Eler, and Andrea Arcuri. 2017. An Empirical Evaluation of Evolutionary Algorithms for Test Suite Generation. In *SSBSE (Lecture Notes in Computer Science)*, Vol. 10452. Springer, 33–48.

[15] Shauvik Roy Choudhary, Alessandra Gorla, and Alessandro Orso. 2015. Automated Test Input Generation for Android: Are We There Yet? (E). In *ASE*. IEEE Computer Society, 429–440.

[16] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evolutionary Computation* 6, 2 (2002), 182–197.

[17] Marcelo Medeiros Eler, José Miguel Rojas, Yan Ge, and Gordon Fraser. 2018. Automated Accessibility Testing of Mobile Apps. In *ICST*. IEEE Computer Society, 116–126.

[18] Phyllis G. Frankl and Oleg Iakounenko. 1998. Further Empirical Studies of Test Effectiveness. In *SIGSOFT FSE*. ACM, 153–162.

[19] Gordon Fraser and Andrea Arcuri. 2011. EvoSuite: automatic test suite generation for object-oriented software. In *SIGSOFT FSE*. ACM, 416–419.

[20] Salvador García, Daniel Molina, Manuel Lozano, and Francisco Herrera. 2009. A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 Special Session on Real Parameter Optimization. *J. Heuristics* 15, 6 (2009), 617–644.

[21] Mona Erfani Joorabchi, Ali Mesbah, and Philippe Kruchten. 2013. Real Challenges in Mobile App Development. In *ESEM*. IEEE Computer Society, 15–24.

[22] Dean C. Karnopp. 1963. Random search techniques for optimization problems. *Automatica* 1, 2-3 (1963), 111–121.

[23] Aravind Machiry, Rohan Tahiliani, and Mayur Naik. 2013. Dynodroid: an input generation system for Android apps. In *ESEC/SIGSOFT FSE*. ACM, 224–234.

[24] Ke Mao, Mark Harman, and Yue Jia. 2016. Sapienz: multi-objective automated testing for Android applications. In *ISSTA*. ACM, 94–105.

[25] Akbar Siami Namin and James H. Andrews. 2009. The influence of size and coverage on test suite effectiveness. In *ISSTA*. ACM, 57–68.

[26] Annibale Panichella, Fitsum Meshesha Kifetew, and Paolo Tonella. 2018. Automated Test Case Generation as a Many-Objective Optimisation Problem with Dynamic Selection of the Targets. *IEEE Trans. Software Eng.* 44, 2 (2018), 122–158.

[27] Annibale Panichella, Fitsum Meshesha Kifetew, and Paolo Tonella. 2018. A large scale empirical comparison of state-of-the-art search-based test case generators. *Information & Software Technology* 104 (2018), 236–256.

[28] Leon Sell, Michael Auer, Christoph Frädrich, Michael Gruber, Philemon Werli, and Gordon Fraser. 2019. An Empirical Evaluation of Search Algorithms for App Testing. In *ICTSS (Lecture Notes in Computer Science)*, Vol. 11812. Springer, 123–139.

[29] Sina Shamshiri, José Miguel Rojas, Gordon Fraser, and Phil McMinn. 2015. Random or Genetic Algorithm Search for Object-Oriented Test Suite Generation?. In *GECCO*. ACM, 1367–1374.

[30] Ting Su, Guozhu Meng, Yuting Chen, Ke Wu, Weiming Yang, Yao Yao, Geguang Pu, Yang Liu, and Zhendong Su. 2017. Guided, stochastic model-based GUI testing of Android apps. In *ESEC/SIGSOFT FSE*. ACM, 245–256.

[31] Hélène Waeselynck, Pascale Thévenod-Fosse, and Olfa Abdellatif-Kaddour. 2007. Simulated annealing applied to test generation: landscape characterization and stopping criteria. *Empirical Software Engineering* 12, 1 (2007), 35–63.

[32] Wenyu Wang, Dengfeng Li, Wei Yang, Yurui Cao, Zhenwen Zhang, Yuetang Deng, and Tao Xie. 2018. An empirical study of Android test generation tools in industrial cases. In *ASE*. ACM, 738–748.

[33] Xia Zeng, Dengfeng Li, Wujie Zheng, Fan Xia, Yuetang Deng, Wing Lam, Wei Yang, and Tao Xie. 2016. Automated test input generation for Android: are we really there yet in an industrial case?. In *SIGSOFT FSE*. ACM, 987–992.