Adaptive Random Testing of Mobile Application*

Zhifang Liu
Department of System Engineering
Beihang University Beijing, China
iuma29@gmail.com

Xiaopeng Gao, Xiang Long
Department of Computer Science and Engineering
Beihang University Beijing, China
{gxp, long}@buaa.edu.cn

Abstract—**Mobile applications are becoming more and more powerful yet also more complex. While mobile application users expect the application to be reliable and secure, the complexity of the mobile application makes it prone to have faults. Mobile application engineers and testers use testing technique to ensure the quality of mobile application. However, the testing of mobile application is time-consuming and hard to automate. In this paper, we model the mobile application from a black box view and propose a distance metric for the test cases of mobile software. We further proposed an ART test case generation technique for mobile application. Our experiment shows our ART tool can both reduce the number of test cases and the time needed to expose first fault when compared with random technique.**

Keywords-component; Adaptive Random Testing; Test Case Generation; Mobile Application

I. INTRODUCTION

The goal of pervasive computing is to realize the dream for everyone to compute and communicate any where any time [6]. Mobile devices are definitely indispensible tools for people to fulfill this dream. Thanks to the advancement of hardware industry, although still resource constrained, modern mobile phones have faster processers, growing memories, faster Internet connections, and much richer sensors. With modern mobile devices, we can make calls to each other, surf the internet, find routes using the built-in GPS, chat or twit with our friends, play games and watch online movies, etc.

However, the rich features we enjoy come at a cost: we need more complex software to support them. The more complex a software is, the more faults and security vulnerabilities it may contain. What is worse, since mobile device is a private person tool, people hope it can be more reliable and secure. Thus, it is the responsibility of mobile application engineers and testers to ensure the quality of mobile applications. Testing is an important tool for mobile application engineers to reduce faults and enhance people's confidence in the mobile application. However, the testing of the mobile application on mobile devices is difficult due to several reasons.

Firstly, the mobile application has to deal with both user inputs and ever-changing environmental contexts. This makes it more complex for testers to generate test cases to expose software faults effectively. To address the first challenge, we model the inputs of mobile application by considering both user inputs and context events.

Secondly, the mobile device is resource limited. If the testing tool or process consumes too much resource, it may affect the normal execution of the mobile application themselves. Thirdly, the diversity of mobile operating systems makes the programming style of different mobile devices differs significantly. Existing white-box testing techniques require either dynamic or static analysis technique to obtain source code information. However, the dynamic analysis is resource demanding as it is performed at runtime while static analysis techniques are highly related to the programming languages used. As a result, it is hard for white-box techniques to be both efficient and general. To be less resource demanding and more general, we abstract the mobile application from a black-box testing view and proposed to use one black-box test case generation technique, i.e., adaptive random testing, for test case generation.

The main contribution of this paper is as follows: (i) It proposes a new model to measure the distance between mobile applications' input represented as event sequence. (ii) It reports the first empirical study on to use such distance on adaptive random test case generation for real-life mobile applications. The results show that our techniques are superior to random test case generation in terms of earlier detection of failures. (iii) It presents other applications of event sequence distance such as the testing other event-driven embedded software.

We organize the rest of paper as follows: Section 2 presents our modeling of mobile application and its inputs. Section 3 first motivate our work and then describes our adaptive random testing algorithm based on event sequence distance. Section 4 presents our empirical study as well as results analysis. Section 5 presents other possible applications of the proposed event sequence distance. Section 6 describes related work, followed by the conclusion in Section 7.

II. MODELLING OF MOBILE SOFTWARE

A. *A black-box view of mobile applicatoin.*

In this paper, we adopt a black-box approach to model mobile application. More specifically, we model a mobile application by observing its input and output information.

From the input perspective, a mobile application receives two kinds of inputs. The first is user GUI input such as keyboard events, touch events. The second is environmental context events which also include two kinds of sources. The

* This research is supported by the National High Technology Research and Development Program of China (project no. 2007AA01Z145).

first kind of context events is the physical contexts obtained from the built-in GPS receiver, Bluetooth chips, accelerometer sensor, humidity sensor, magnetic sensor and network, etc. The second kind of context events is the social contexts like nearby MSN friends, the current activity the user is up to, and even the user's mood. The mobile application must accept and react to both kinds of continuously changing context events as inputs to produce appropriate outputs.

From the output perspective, the mobile application generates multiple mode outputs like display image, video, audio, vibration, LED signals. So the output of a mobile application is a superset of traditional GUI application.

B. Modelling the Inputs of mobile application

In [1], Memon et al. proposed to model GUI applications with event-flow graph. They define a GUI as a hierarchical, graphical front-end to a software system that accepts as input user-generated and system-generated events from a fixed set of events and produces deterministic graphical output. They use event-flow graph to represent all the possible interaction of different GUI events. However, their models are not suitable for modeling the non-determinism caused by the interleaving of dynamic context updates.

In [2], Michele Sama et al. proposed to model mobile context-aware adaptive application with a state graph for model checking. Their technique can effectively model the non-determinism caused by the interleaving of context updates. However, they only consider context events within their model, so their model is only suitable for modeling mobile application driven purely by events.

In this paper, we are targeting at general mobile applications whose inputs contain not only GUI events but also context events. We propose to model the inputs of mobile application as a sequence of events in which each event can be either a GUI event or a context event.

We represent each event e as a pair <t, v> where t represent the event type of the corresponding event and v represent the value of the corresponding event. We further define a test case T as:

T = <e1, e2, … , en> where each ei = <ti , vi>.

The event type represents different kinds of events from various event sources. We list typical events consumed by a mobile application in TABLE I. The column 'Event Type' represents different type of events. The column 'Example' gives an example of the possible value of an event. Finally, the column 'Alphabet-Number Shorthand' uses an alphabet or number shorthand to represent the corresponding event type. We will use the shorthand for calculating the event type distance as discussed in next section.

For example, we can interpret the first row of the table as follows. A key is pressed by user which triggers an event of type 'Key Pressed' with key value equal to '1'. Similarly, we can interpret the row six as follows: the Bluetooth is active which triggers an event of type 'Bluetooth' and a Boolean value 'TRUE'.

The value v of different event type t can have different format. For example, the value for event type 'Key Pressed' is an alphabet or number representing the value of the key pressed. The format for the value of event type GPS is a pair of number resenting its longitude and latitude. While the format for the value of user activity is a string representing the activity description.

TABLE I. TYPICAL INPUT EVENTS FOR MOBILE APPLICATION

Event Type	Example	Alphabet-Number Shorthand
Key Pressed	Press key '1'	0
Key Released	Release key '1'	1
Click	Click on widget 'Gmail'	2
Long Click	Long click widge "Dial"	3
Track Ball Move	Track ball move 'left'	4
Bluetooth	Bluetooth 'enabled'	5
WiFi	WiFi 'enabled'	6
GPS	GPS data (80E, 30N)	7
Humidity	Relative Humidity 70%	8
Accelerometer	Speed up at 5km/h	9
Magnetic Data	Magnetic field value 50	A
User Activity	In meeting	B
Phone Usage	Inactive	C

III. ADAPTIVE RANDOM TESTING OF MOBILE APPLICATION

A. Motivation

The testing of mobile applications in industry is still far from fully automatic. The testing engineers generally use two approaches to automate their time-consuming work. The first one is the capture and replay tool with which the testers can automatically record test scripts by manipulating the mobile application under test. Then the testers can replay the recorded test scripts repeatedly afterwards. The problem with record-replay technique is that the quality of the generated test scripts depends on the testers' familiarity and understanding of the application under test, which can fluctuate greatly across different testers and applications. Another technique is for the testers to write test scripts directly to manipulate the application under test. However, the development of high quality test scripts is nontrivial and quite time consuming. Thus, an automatic and effective test case generation technique for mobile application is highly desirable.

Random testing can be fully automatic, non-intrusive. For example, there is a built-in Monkey application within the Android mobile OS. Testers can ask the Monkey to send random event sequences targeted at a specific application. However, pure random testing, although simple and fully automatic, may not be effective for detecting a fault.

Researchers have proposed cleverly designed random testing technique to improve its effectiveness. Adaptive Random Testing is one such cleverly designed random testing technique for test case generation. Based on the

observation, the input failure regions of an application tend to cluster together, it tries to spread the randomly generated test cases as evenly as possible. Empirical studies show ART can reduce the number of test cases needed to find the first fault by as much as 40% to 50% [4]. The seminal works on ART focus on numerical application [4][5]. Since ART relies on the definition test case distance based on input information, extending the ART idea on application with different input format is a research question by itself. In [7], Ciupa et al. propose the definition of object distance and they further extend ART for object-oriented software. In this paper, we want to extend the Adaptive Random Testing to the automatic test case generation for mobile application. The inputs for mobile application are event sequences and there no existing works on this aspect. In this paper, we first propose the definition of test case distance for mobile applications and we further adapt the ART algorithm to generate test cases automatically for mobile applications.

B. The ART Algorithm

We propose the following ART algorithm which is adapted from the FSCS-ART proposed by Chen et al. [4][5] for testing mobile application.

```
T = { } /*T is the set of previously executed test cases*/
randomly generate an input t
test the program using t as a test case
add t to T
while(no fault is detected)
D = 0
    randomly generate k candidates c1, c2,…, ck
       from event   pools
for each candidate ci
  calculate the distance di with its nearest neighbor in T
  if di > D
      D = di
        t = ci
   end if
end for
add t to T
test the program using t as a test case
end while
```

Figure 1. The Adaptive Random Testing Algorithm

Similar to the FSCS-ART algorithm, each time we select a test case that is farthest away from its nearest neighbor in the already executed test cases.

Unlike the FSCS-ART algorithm, we use a different approach to generate the set of candidate test cases. Since a test case is a sequence of events of different types. We first generate an event pool for each kind of events. To generate an event of an event sequence, we first randomly select an event pool and then we randomly sample an event from the pool, we repeat this process until an event sequence of length l is generated. The length l is the average events number received by an application during a user session obtained from the usage profile history for an application. We repeat this process until we have generated k candidate event sequences. The candidate set size k is a constant obtained empirically. Both the random selection of event pools and the random selection of events within a specific event pool make the generated candidate test case more diversified.

C. Test Case Distance for Mobile Application

One key aspect of the ART algorithm is to define the distance between test cases. In our model, A test case for mobile application is an ordered list of different type of events, which in turn have different values. Thus, to measure the distance between two event sequences, we must take consider into the following two components:

- Sequence distance: a measure of the distance between two sequences of events of various types, without considering their values.
- Value distance: a measure of the distance between two event sequences that have the same event type for each corresponding event.

Thus, we can define the distance between two event sequences es_i and es_i as:

distance (es_i, es_j) =
(*norm* (**sequence_distance** (es_i, es_j))
+ *norm* (**value_distance** (es_i, es_j))) /2

In the distance definition, *norm()* represents a non-decreasing function to normalize the corresponding distance within the range from 0 to 1. We now look at the two components of the distance.

The sequence distance is a measure of the distance between two ordered lists of event types. Since we only consider the type of an event rather than the value, we can represent each event type as an alphabet-number character as shown in the third column of TABLE I. Thus, we can donate each event sequence as a string composed of Alphabet-Number characters corresponding to certain event type. As a result, the problem of defining sequence distance is transformed to the problem of measuring the distance between two strings. In this paper, we adopt one of the standard string distance metric: Levenshtein distance [3]. We use *char*(es_i) to represent the character encoding of a event sequence es_i. We define sequence distance as:

sequence_distance (es_i, es_j) =
Levenshtein (*char*(es_i), *char*(es_j))

Since the Levenshtein distance measures the minimum edit operation needed to transform on string to another. We also obtain the transformed string for measuring the value distance. For example, suppose the Levenshtein algorithm transforms the string *char*(es_i) to string *char*(es_j) with some basic edit operations. This corresponding to add or remove some events within the event sequence es_i *to get a* transformed sequence *trans*(es_i). Note that for the addition of new events, we also initialize the event with a default value for measuring value distance later on. Since the transformed event sequence *trans*(es_i) has exactly the same type and length with the event sequence es_j, we can calculate their value distance by sum up all distance between the value of their corresponding events and then perform normalization.

value_distance (es_i, es_j)
= **value_distance** (*trans*(es_i), es_j)

$$= norm\left(\sum_{m=1}^{|trans(es_i)|} distance\left(trans(es_i)[m], es_j[m]\right)\right)$$

Finally, we must define *distance*(e_i, e_j), the elementary distance between two event values of same type. For two events e_i and e_j, this definition varies for different event types.

- For numbers, their Euclid distance normalized to range [0,1].
- For high dimensional tuples, the Euclid distance between them normalized to range [0,1].
- For Booleans, 0 is identical, 1 otherwise.
- For strings, the Levenshtein edit distance.

In the following section, we will use the defined event sequence distance to propose an adaptive random test case generation technique.

IV. EVALUATION

In this section, we perform an experimental study to evaluate the effectiveness of our ART test case generation technique.

A. Research Questions

RQ1: Is ART more effective than random by reducing the number of test cases required to expose first fault (F-measure) to test mobile application?

RQ2: Can ART reduce the time needed to find first fault than random to test mobile application?

B. Subject Programs

As shown in TABLE II. , we use six real-life applications running on Android 1.5 Mobile OS to evaluate our ART test case generation technique. The Contact, Dialer, and Brower application are built-in applications of a custom build OS ROM. While the Bluetooth, SMS, Bluetalk applications are third-party applications.

TABLE II. SUBJECT PROGRAMS

Application	Description
Bluetooth	A Bluetooth application for file sharing
Contact	Management of user contact
SMS	SMS client for sending and receiving SMS
Bluetalk	An VOIP Bluetooth application
Dialer	Make and answering calls
Browser	Mobile Web Browser for surfing Internet

C. Experimental Environment

We implement our ART test case generation algorithm as a tool called smart-monkey within MobileTest [8], a testing framework for automatic black box testing of mobile applications. MobileTest originally uses capture-replay mechanism to help testers automate their test case generation task. With the smart-monkey tool, MobileTest now supports fully automatic adaptive random testing of mobile applications. Our experiment is performed on a PC having a Pentium 3.0 GHz processor, 4 GB of RAM, and running Windows XP SP2.

D. Effectiveness Metrics

We use two effectiveness metrics. The first one is *F-measure* [8], the *number of test cases required to detect the first failure*. It reflects the fault detection ability of the generated test cases. The second one is the *time used to find the first fault*. This second metric also takes the time used for test case generation into consideration.

E. Experiment

We applied both the random strategy and ART to test one application at a time. For the experiment setup, we first generate a pool for each kind of events. The pool ensures each generated event has valid value for the application to consume. For the random strategy, each time it randomly sample from the event pools to generate a event sequence (test case) and pass it to the application under test. For ART strategy, each time we generate a new test case use the algorithm presented in section III. For both strategies, we repeatedly generate new test cases until the first fault is exposed. To eliminate the impact of pseudo-random generator, we repeat the testing process 10 times with different seeds and average out the results. Our tool exposes a fault if a software crash or a system no response error is triggered.

F. Results and Analysis

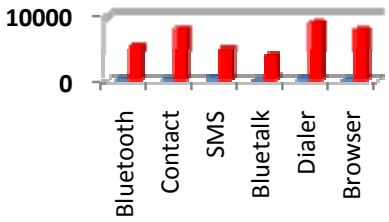

Figure 2. Comparison of F-measure between ART and Random

We compare the F-measure (number of test cases to first fault) between random and ART techniques as shown in . The x-axis represents different applications and the y-axis represents the number of test cases required to detect the first fault. We can see that across all programs, ART test case generation technique use much less test cases than random to detect the first fault. This shows that evenly spread the event sequences can increase the fault detection ability of the generated test cases.

Although ART can reduce the number of test cases required to detect first fault, it takes more time than random technique to generate the test cases. So we continue to compare the time needed to find the first fault between ART and random as shown in . The x-axis represents different applications and the y-axis represents the time needed to expose the first fault. Note that, the time measured here includes both the test case generation time and test case execution time. We can see that ART still use less total time than random to expose the first fault across all application. After considering the test case generation time, the improvement of ART over random is not as drastic as F-measure, but it is still significant. For

example, the ART use around 500 second less time than random to expose the first fault for the Contact application.

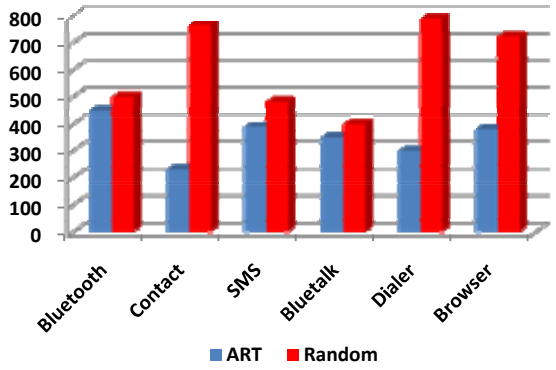

Figure 3. Comparison of time to first fault between ART and Random

From the results of the experiment, we can answer the research questions raised at the beginning of this section: Compared with Random, Adaptive Random Testing can both reduce the number of test cases and the time needed to expose the first fault significantly.

V. Further Discussion

Our event sequence distance and ART test case generation technique are not limited to the testing of mobile applications. In fact, it is applicable to a broad range of embedded event-driven software. For example, our tools can test the set-top box system for interactive TV, the play station for video games and sensor-driven automaton arms as long as the input events are defined. We have performed a case study to use our smart monkey tool for testing set-top boxes. We found it could reduce the time to find faults in top-set system by 40% than random.

VI. Related Work

In [1], Memon et al. proposed to model GUI applications with event-flow graph. Then they perform systematic test case generation over the graph model to the testing of GUI applications. However, their technique requires detailed static or dynamic analysis of GUI structure. The static analysis requires source code while the dynamic analysis requires non-trivial overhead during program execution. On the contrary, our ART test case generation technique is a black box, lightweight technique suited well for resource constrained mobile applications.

Adaptive random testing [4][5] improves the performance of random testing by carefully guide the test case generation process. Ciupa et al. investigated how to define the distance among objects for ART [7]. Their experimental results show that ART based on object distance can significantly increase the fault detection rate for object-oriented programs

VII. Conclusion and Future Work

Mobile applications are driven by GUI and context events. In this paper, we proposed the event sequence distance to measure the distance between the test cases of mobile applications. We further proposed the ART test case generation technique to generate test cases for mobile application in a black box, non-intrusive manner. Our experimental results show that our ART test case generation technique can both reduce the number of test cases and the time required to expose first failure when compared with random technique.

In future work, we plan to combine ART with model based testing: we want to use ART strategy to explore the model space of mobile application as evenly as possible. We also want to combine ART with profile guided test case generation, i.e., we want the generated test case to be similar to the profiles of human testers while still diversified among themselves.

References

[1] A. M. Memon and Q. Xie, "Studying the fault-detection effectiveness of GUI test cases for rapidly evolving software". IEEE Transactions on Software Engineering 31, 10 (Oct.), 884–896, 2005.

[2] M. Sama, D. S. Rosenblum, Z. Wang, and S. Elbaum, 2008. "Model-based fault detection in context-aware adaptive applications," In Proceedings of the 16th ACM SIGSOFT international Symposium on Foundations of Software Engineering (FSE-16).. ACM, New York, NY, 261-271.

[3] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," Doklady Akademii Nauk SSSR 163, 4 (1965), 845–848.

[4] T. Y. Chen, F.-C. Kuo, R. G. Merkel, and T. H. Tse. "Adaptive random testing: the ART of test case diversity," Journal of Systems and Software, 2009. doi:10.1016/j.jss.2009.02. 022.

[5] T. Y. Chen, H. Leung, and I. K. Mak. "Adaptive random testing," In Advances in Computer Science: Proceedings of the 9th Asian Computing Science Conference (ASIAN 2004), volume 3321 of M. J. Maher (ed.), Lecture Notes in Computer Science, pages 320–329. Springer, Berlin, Germany, 2004.

[6] M. Weiser, "Some computer science issues in ubiquitous computing," SIGMOBILE Mob. Comput. Commun. Rev. 3, 3, 12, 1999.

[7] I. Ciupa, A. Leitner, M. Oriol, and B. Meyer. "ARTOO: adaptive random testing for object-oriented software," In Proceedings of the 30th International Conference on Software Engineering (ICSE 2008), pages 71–80. ACM Press, New York, NY, 2008.

[8] B. Jiang, X. Long, and X. P. Gao. MobileTest: A Tool Supporting Automatic Black Box Test for Software on Smart Mobile Devices. In Proceedings of the Second international Workshop on Automation of Software Test (AST 2007). IEEE Computer Society, Washington, DC, 8, 2007.

[9] Chen, T. Y. and Kuo, F. 2006. "Is adaptive random testing really better than random testing," In Proceedings of the 1st international Workshop on Random Testing (Portland, Maine, July 20 - 20, 2006). RT '06. ACM, New York, NY, 64-69.