Journal of Universal Computer Science, vol. 29, no. 2 (2023), 152-178
submitted: 8/5/2022, accepted: 21/11/2022, appeared: 28/2/2023

An Enhanced Testing Approach for Mobile Applications

Amira Samir
(Department of Information Systems, Faculty of Computer and Information Sciences, Ain Shams University, Cairo, Egypt
https://orcid.org/0000-0003-4641-0585, amira_samir@cis.asu.edu.eg)

Huda Amin Maghawry
(Department of Information Systems, Faculty of Computer and Information Sciences, Ain Shams University, Cairo, Egypt
https://orcid.org/0000-0001-5550-5717, huda_amin@cis.asu.edu.eg)

Nagwa Badr
(Department of Information Systems, Faculty of Computer and Information Sciences, Ain Shams University, Cairo, Egypt
https://orcid.org/0000-0002-5382-1385, nagwabadr@cis.asu.edu.eg)

Abstract: Nowadays, there is an enormous number of mobile applications that are continuously being launched to the market. As a result of this rapid process, there is a need to increase the speed of testing process using enhanced approaches. This research aims to increase the effectiveness of the graphical user interface testing process of mobile applications. This is achieved by proposing an enhanced combinatorial-based metaheuristic approach. The proposed approach aims to maximize statement and branch coverage by applying Cuckoo search, for event selection. The approach was compared to monkey, frequency, random and greedy approaches. Experiments were conducted on different mobile applications. During the same testing time duration, the proposed approach achieved higher coverage than the other approaches. The proposed approach proved its effectiveness in mobile application testing compared to the other approaches.

Keywords: combinatorial testing; metaheuristic; mobile application testing; automated testing
Categories: D.2, D.2.5
DOI: 10.3897/jucs.86295

1 Introduction

Nowadays, mobiles have become part of the entire community. Mobile users are more than 3 billion around the world [S. O'Dea, 2020]. Currently, millions of mobile applications are launched to the market [Clement, 2020].

One of the important issues related to those applications is testing to ensure their quality before publishing [Kaur, 2015]. Automated testing is used for increasing quality of an application and decreasing the effort and time [Dustin et al., 2009].

One type of testing is graphical user interface (GUI) testing. GUI consists of a large percentage of an application's code [Memon, 2002]. GUI testing verifies application screen components and tests the user's interaction through them [Karthikeyani, 2011, Naik and Tripathy, 2011]. GUI testing is important to ensure the functionality of the applications.

There are different testing approaches such as Monkey ['UI Application Exerciser Monkey - Android Developers', 2020]. Monkey testing is based on generating random events. Combinatorial testing (CT) is another approach that is based on combining parameters and testing the interactions between them to find errors [Ince et al., 2014, Kuhn et al., 2010]. The parameters could be the testing events [Adamo et al., 2018b, Samir et al., 2019]. A collection of appended events is considered as a test case. A group of associated test cases files is considered as a test suite [Homès, 2011].
Cuckoo search [Yang and Deb, 2009] is a metaheuristic algorithm. It is used for selecting the most suitable event to be executed and added to a test case by finding the best set of solutions that has the smallest fitness value and selecting the event that has the nearest magnitude that satisfies the value of a solution from the best set of solutions. The proposed approach applies enhancements to Cuckoo search and is evaluated in terms of statement and branch coverage. Statement coverage indicates that each statement included in the code is executed by test suite [Homès, 2011]. Branch coverage indicates that each decision output included in the code is executed by a test suite [Homès, 2011]. Therefore, the generated test cases were evaluated using two metrics, statement, and branch coverage to avoid any bias could happen when applying only one metric. For maximizing the statement and branch coverage of the generated test cases, a combinatorial-based metaheuristic approach is proposed by applying Cuckoo search for event selection.

Compared to other approaches such as monkey ['UI Application Exerciser Monkey - Android Developers', 2020], random [Adamo et al., 2018b], frequency [Machiry et al., 2013] and greedy [Adamo et al., 2018b] approaches, the proposed approach outperformed them.

The main contributions of this paper are:

Proposing an automated mobile application GUI testing approach.

Applying enhancements to Cuckoo search approach.

Enhancing statement and branch coverage compared to monkey, random, frequency and greedy approaches.

The proposed approach is a black-box approach. It aims at maximizing statement and branch coverage and generating test suites without redundancy in selecting events. It also aims to prove that the proposed metaheuristic approach is more effective than greedy approach for combinatorial mobile application testing.

The paper is organized as follows: background about Cuckoo Search is presented in section 2. Then, related studies are discussed in section 3. In section 4, the proposed approach is explained with its system architecture followed by experiments and results in section 5. Finally, the conclusions of this paper and future work are stated.

2 Background

Cuckoo search is inspired from Cuckoos [Payne, 2005]. They are birds with a strange way in raising their offspring as they depend on other species of birds. A female cuckoo bird puts her eggs in a different species nest and removes the available eggs. A host could detect that there is a strange egg at its nest, so it could drop the egg or neglect this nest and start constructing a new nest. Some species of cuckoos could hatch before their hosts' eggs, that the host thinks that a cuckoo egg is its own chick. By this way, young cuckoos could be raised by the host species.

Cuckoo search (CS) [Yang and Deb, 2009] is a metaheuristic algorithm based on the strange attitude of cuckoo birds. Each cuckoo bird lays in a random nest (set of solutions) an egg (a solution). Upper and lower bounds could be used for a uniform random distribution, making solutions between them [Yang, 2010]. The probability that the host discovers cuckoo's solution ranges from 0 to 1, that those set of solutions are discarded, and new ones are built instead.

The algorithm iterates till the termination criterion is achieved. Each iteration, a new set is generated with random solutions and its fitness is calculated (F_m). Then a random set of solutions is selected from the list of available sets of solutions and its fitness is calculated (F_k). If the new set's fitness is smaller, then set the new solutions to replace the set at index k. Then, some sets are discarded according to the probability p_a. New sets are built to replace the discarded sets. Then, the fitness of each new set is calculated. After that, the sets of solutions are sorted ascendingly according to their fitness value. The best set is selected, which is the set with the minimum fitness. The Cuckoo search (CS) pseudocode is shown in *Algorithm 1.*

Algorithm 1: Cuckoo Search [Yang and Deb, 2009].

Input: number of sets of solutions, p_a

1: **Begin**
2: Set fitness function $f(x)$
3: Set starting population of z set of solutions x_m, $m = 1$ to z
4: **While** (termination criterion or maximum iterations count have not been achieved) **Do**
5: Create a random set of solutions
6: Calculate its fitness with F_m fitness function
7: Pick a random set of solutions (k)
8: Calculate fitness of the selected random nest (F_k)
9: **If** $F_m < F_k$ **Then**
10: Set new solution to the set (k)
11: **End If**
12: Discard a part p_a of worst sets of solutions
13: Generate new sets of solutions
14: Evaluate fitness of new sets of solutions
15: Sort solutions
16: Find best solutions
17: **End While**
18: **End**

3 Related Work

The following subsections present an overview about recent relevant approaches including automated mobile application testing, Cuckoo search testing, combinatorial testing, and combinatorial mobile application testing. The following subsections go as follow: section 3.1 introduces several studies about automated mobile application testing. Section 3.2 discusses several studies related to Cuckoo search approach and its applications in testing. Section 3.3 presents several combinatorial testing studies. Finally, section 3.4 introduces several studies that utilized combinatorial testing in automated mobile application testing.

3.1 Automated Mobile Application Testing

Several studies about mobile application testing using automated approaches are introduced in literature. Monkey ['UI Application Exerciser Monkey - Android Developers', 2020] is a tool that generates automated random events. Random approach generates relevant random events to the running screen of application. Machiry et al. [Machiry et al., 2013] presented three approaches in Dynodroid for selecting an event by using the history of number of executions of all events. The proposed approach is compared to frequency approach which selects the minimum executed event. Arnatovich et al. [Arnatovich et al., 2018] have presented Mobolic which is an automated testing approach. They have built a Finite-state graphical user interface flow graph (F-GFG) model for the GUI of an application. Then they have used the A* search algorithm [Hart et al., 1968] for exploration of this model to find the shortest path. They have also generated customized inputs. Salihu et al. [Salihu et al., 2019] have proposed a tool that extracts mobile application's events by a static analyzer to generate a model then crawls dynamically using Dijkstra's algorithm to find the shortest path through it and generate test cases. Ferreira et al. [Ferreira and Paiva, 2019] have built a Finite State Machine (FSM) of an application's GUI. They explored it using Dijkstra's algorithm based on a specific defined priority for events. Li et al. [Li et al., 2019] have presented a testing tool based on deep learning. They have trained the Rico dataset [Deka et al., 2017] to build the user interactions model that will be used. Their target is to generate inputs as human testers to increase coverage.

3.2 Cuckoo Search Testing

Some authors have used Cuckoo search for test cases optimization. Cuckoo search approach can minimize the number of the generated test cases. For example, Ahmed [Ahmed, 2016] has proposed an approach for optimizing the set of test cases using CS. CS finds the test case that covers more tuples from the combinations list that have been previously generated. Khan et al. [Khan et al., 2018] have developed a technique for generating optimized test cases for path testing by combining CS and genetic algorithm. CS is used for filtering the previously random generated test cases. Sharma et al. [Sharma et al., 2019] have applied CS in data flow testing. First, they generated control flow graph (CFG) and definition use path, then applied Cuckoo search for generating optimized test cases.

Moreover, CS has been used for test data optimization. Khari et al. [Khari and Kumar, 2017] have proposed an approach applying CS for generating an optimized test

data automatically. Their approach generated a great amount of test data, then test data was ensured that it was relevant to the tested path and CS was applied to check whether to keep or remove test data from the list.

For test cases prioritization, Bajaj et al. [Bajaj and Sangwan, 2021] have proposed hybrid approach merging CS with genetic algorithm. They have focused on prioritizing test cases for regression testing.

CS is applied in this paper for GUI testing. The proposed approach generates test suites with maximized statement and branch coverage by avoiding redundancy in selecting the events that would be added to test cases.

3.3 Combinatorial Testing

Some studies applied combinatorial-based testing approaches. Bombarda et al. [Bombarda and Gargantini, 2020] have proposed a combinatorial testing approach that generates test sequences through covering the interactions that happen at a software represented as Finite State Machines (FSM). Alsewari et al. [Alsewari et al., 2020] have proposed a combinatorial testing approach based on the Harmony search algorithm [Geem et al., 2001] for effectively exploration by controlling the use of previous solutions and controlling randomness. Zamli et al. [Zamli et al., 2020] have enhanced the Sine Cosine algorithm (SCA) [Mirjalili, 2016] for optimizing the generated combinatorial test suites. Richter et al. [Richter et al., 2020] have enhanced the Avocado testing tool ['Avocado Testing Framework', 2020] by adding a plugin for combinatorial testing.

3.4 Combinatorial Mobile Application Testing

For automated mobile application testing, multiple studies have proposed combinatorial-based approaches. Adamo et al. [Adamo et al., 2018b] have proposed a combinatorial testing approach by applying greedy algorithm for selecting events. The combinatorial-based approach with combination strength of two got higher statement coverage than random, frequency [Machiry et al., 2013] and monkey. Samir et al. [Samir et al., 2019] have proposed an enhanced combinatorial testing approach based on [Adamo et al., 2018b] by applying Last In First Out (LIFO) tie-breaking and adding weight to events in order to select the event that should be executed. It is compared to First In First Out (FIFO) and random tie-breaking. Huynh et al. [Huynh et al., 2019] have proposed a combinatorial approach based on In-Parameter-Order (IPO) algorithm for generating test cases and test data used for creating unit tests with JUnit. Michaels et al. [Michaels et al., 2020] have presented a combinatorial approach for reducing test suite's size with the minimum loss of code coverage.

Drawbacks of the existing automated mobile application and combinatorial mobile application testing approaches are presented at Table 1. Those drawbacks have been avoided at the proposed approach. The proposed approach has been evaluated through ten applications under test (AUTs). Moreover, it generates relevant test cases with maximized statement and branch coverage.

Therefore, this paper proposed an efficient automated mobile application testing approach. It is a combinatorial testing approach that applies metaheuristic Cuckoo search algorithm. Cuckoo search is used for selecting the most suitable event to be

executed and added to a test case. The proposed approach prevents redundancy in selection. It generates relevant events. This leads to having a test suite with a maximized statement and branch coverage compared to four existing approaches which are monkey, random, frequency and greedy. Some existing approaches could need more evaluation as they are evaluated based on one application. The proposed approach has been evaluated using ten applications. The time duration of test cases generation is an input so there is no chance to take long time for execution.

Approach	**Drawbacks**
Monkey ['UI Application Exerciser Monkey - Android Developers', 2020]	Generation of non-related events
Random [Adamo et al., 2018b]	Same event could be selected multiple of times
Frequency [Machiry et al., 2013]	Long time to complete execution
Greedy [Adamo et al., 2018b]	Applying random tie-breaking for selecting event
[Arnatovich et al., 2018]	Incomplete exploration of UI
[Salihu et al., 2019]	Not handling events between applications
[Ferreira and Paiva, 2019]	Not compared to existing approaches
[Li et al., 2019]	Unreliable coverage for complex applications
[Samir et al., 2019]	Evaluated using a few numbers of AUTs
[Huynh et al., 2019]	Evaluated using one AUT only
[Michaels et al., 2020]	Evaluated using a few numbers of AUTs

Table 1: Drawbacks of existing automated mobile application and combinatorial mobile application testing approaches

4 Proposed Approach

An enhanced automated testing approach for mobile application GUI is proposed. It is a combinatorial-based metaheuristic approach for maximizing statement and branch coverage using Cuckoo search algorithm [Yang and Deb, 2009]. The architecture of the proposed approach consists of three layers which are user interface layer, testing layer and data layer as shown in Figure 1.

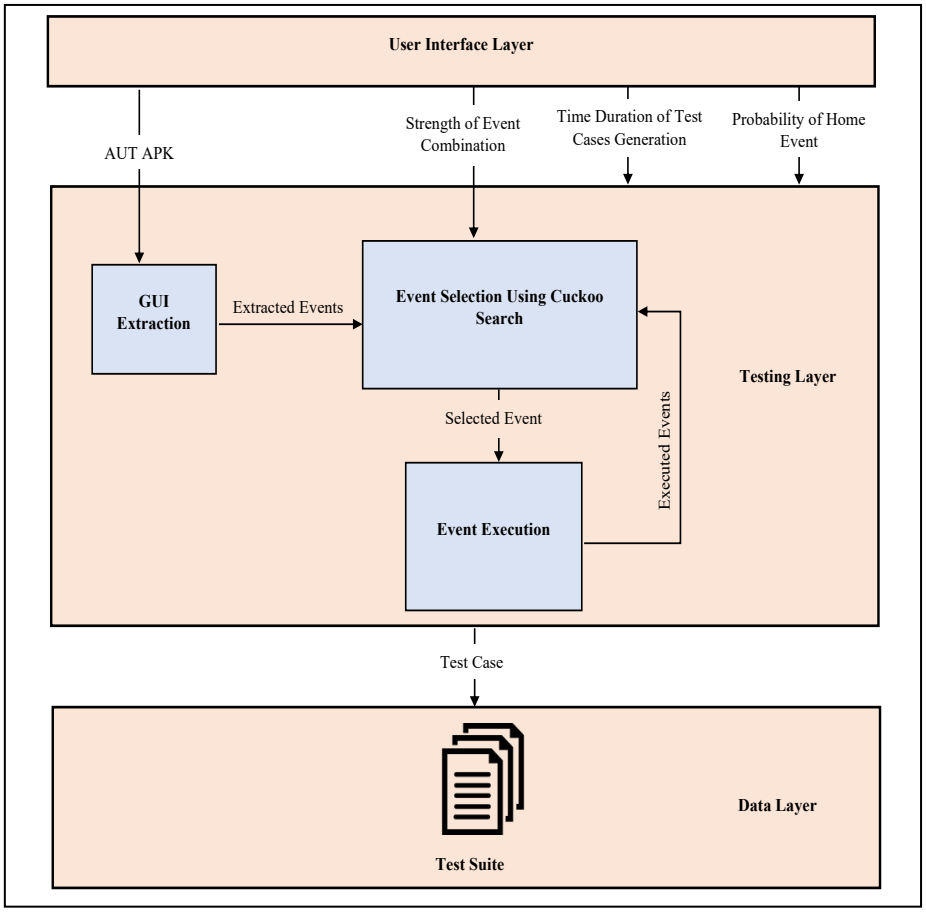

Figure 1: Proposed Testing Approach Architecture

4.1 User Interface Layer

User interface layer presents the front end where the user could insert the Application Under Test Android Application Package (AUT APK). It is a black-box testing approach based on the AUT's requirements that there is no necessity for the presence of source code of applications for generating test cases. It requires AUT APK only. Besides, the user could insert the required information that will be used for the testing

process including the time duration of test cases generation, probability of home event and strength of event combination. Probability of home event is the likelihood of the selection of home event. It ranges from 0 to 1 [Lavrakas, 2008]. It is used to terminate a test case, as an AUT could be closed by executing home event. The time duration of test cases generation is used to terminate the run to generate one test suite. Strength of event combination n indicates finding all possible combinations of events with this number (n-way) [Bryce et al., 2009, Kuhn et al., 2013].

4.2 Testing Layer

Testing layer consists of three modules which are GUI extraction, event selection using Cuckoo search and event execution. First, the testing time duration is checked to see whether to terminate the testing process or not. The process would iterate till the testing time duration exceeds the time that the user has entered. Then, a random generated value is compared to the probability of home event. When this value is greater than the probability of home event, then home event would be executed, and the test case generation process would terminate. The output of this module is a single test case.

4.2.1 GUI Extraction

GUI extraction module extracts the GUI hierarchical representation of AUT screens. It finds all the possible executable events using Appium [Appium, 2020]. Appium is recognized as an open-source software for automated testing of mobile applications. The output will be a list of events that would be used by the next module.

4.2.2 Event Selection Using Cuckoo Search

This is the main module at the testing layer. This module uses the stored extracted events list and the strength of event combination as inputs. The output of this module is a selected event that would be executed by the next module. The Pseudocode is shown at *Algorithm 2.*

Algorithm 2: Event Selection Using Cuckoo Search.

Input: List of extracted events, list of executed events, strength of event combination n
Output: Selected event

1: **Begin**
2: Get weights of all events
3: Set upper bound to maximum weight
4: Set lower bound to minimum weight
5: Set number of solutions in each set which is n
6: Set number of sets of solutions and p_a
7: **While** (number of iterations have not been achieved) **Do**
8: Generate a random set of solutions with upper and lower bounds using Lévy flights

Algorithm 2: Event Selection Using Cuckoo Search.

```
9:      Calculate its fitness with the Michalewicz's function F_m
10:     If solutions inside the set are previously executed Then
11:         Increase fitness value
12:     End If
13:     Select a random set of solutions (k)
14:     Calculate fitness of the selected random set of solutions (F_k)
15:     If F_m < F_k Then
16:         Place new solution to the set (k)
17:     End If
18:     Discard a part p_a of worst sets of solutions
19:     Generate new sets of solutions
20:     Evaluate fitness of new sets of solutions
21:     Sort solutions ascending
22:     Find best set of solutions with the smallest fitness
23: End While
24: Retrieve the event with the nearest magnitude of weight of all solutions in the best set
25: Select this event
26: End
```

A. Starting Population:

The weight of each event is computed using the following equation [Adamo et al., 2018a] :

$$\text{weight} = \frac{1}{1+N} \tag{1}$$

where N is the number of previous selections of an event. Then, based on (1), the maximum weight of events list is set for upper bound, and the minimum weight is set for lower bound. Those bounds are used for generating random solutions within range when applying Cuckoo search algorithm. Strength of event combination determines the number of solutions in each set. A solution is a random value calculated based on upper

and lower bounds. Then, the number of sets of solutions, p_a and the termination criterion have been defined.

B. Sets of solutions:

The number of sets of solutions is defined as 10 times the number of extracted events. This is the most optimal value, as on increasing this number the performance of the approach decreases, by taking too much time when generating a test case, resulting in minimizing the statement and branch coverage. However, on decreasing this number, that decreases their variety and the opportunity of finding out the best set of solutions. The probability that the sets of solutions are discarded is $p_a \in [0,1]$, then generating new sets instead.

C. Termination Criterion:

Then, there are iterations generating sets of solutions and finding the set with the smallest fitness till it reaches the termination criterion. Fitness value is calculated using Michalewicz's function in (7). In the end, the event with the nearest magnitude of weight of all solutions in the selected set is chosen to be executed. The termination criterion is defined as the maximum iterations count which is 850. This is the most optimal value, as decreasing this value minimizes the execution time of the Cuckoo search approach, resulting in inability to reach the optimal set of solutions. However, on increasing this value, the approach is taking too much time when generating a test case, minimizing the statement and branch coverage. If the maximum value is reached, then the best set of solutions is returned. If not, then a new set is generated with random solutions within range of bounds using Lévy flights in (2).

D. Lévy flights:

In Cuckoo Search, a new set of solutions is produced with a random step by applying Lévy flight.

$$x_i^{(n+1)} = x_i^{(n)} + \alpha \oplus l\acute{e}vy\ (\lambda) \tag{2}$$

Lévy flights are calculated based on (3) and (4). Where n is the iteration, α is the length of a step factor, $l\acute{e}vy\ (\lambda)$ represents the random walks obtained by Lévy flight and $\oplus$ represents the entry-wise multiplication. λ is exponential coefficient [Zhang et al., 2020], it is within $(1 < \lambda \leq 3)$.

α is calculated using the following equation:

$$\alpha = \frac{L}{100} \tag{3}$$

where L is the length scale [Yang and Deb, 2010].

For generating the steps [Yang, 2010], Mantegna's algorithm [Mantegna, 1994] is used, where the length of step which is $l\acute{e}vy\ (\lambda)$ is calculated as in the following equation:

$$l\acute{e}vy\ (\lambda) = \frac{c}{|d|^{\frac{1}{\lambda}}} \tag{4}$$

where c and d are random parameters extracted from normal distributions based on (5) and (6).

$$C \sim N(0,\sigma_c^2),\ d \sim N(0,\sigma_d^2) \tag{5}$$

where

$$\sigma_c = \left(\frac{\Gamma(1+\lambda)\sin\left(\frac{\pi\lambda}{2}\right)}{\Gamma\left(\frac{(1+\lambda)}{2}\right)\lambda 2^{\frac{(\lambda-1)}{2}}} \right)^{1/\lambda}$$

$$,\ \sigma_d = 1 \tag{6}$$

where $\Gamma(z)$ denotes the gamma distribution function [Artin, 1964], and z is a complex number. Most of the researchers set $\lambda = 1.5$ and $p_a = 0.25$ [Yang and Deb, 2010].

E. Fitness Function:

Fitness of each set of solutions is calculated using Michalewicz's function in (7) [Michalewicz, 1992, Molga and Smutnicki, 2005]:

$$f(x) = -\sum_{i=1}^{n} \sin(x_i)\left[\sin\left(\frac{ix_i^2}{\pi}\right)\right]^{2m} \tag{7}$$

It is a multimodal function where there are several local minima. m has been set to 10, where m defines the steepness parameter.

After calculating fitness for each set of solutions, it is important to ensure that those solutions have not been selected and executed before. The set's fitness value has been increased according to its solutions. Then, a random set of solutions (k) is selected, and its fitness is calculated. Then, the fitness of the selected random set (F_k) and the new generated set (F_m) are compared, if the new set of solutions' fitness is smaller, then replace the set at index (k) with new solutions.

Then, some sets are discarded according to the probability p_a, and new sets are generated using Lévy flights to replace them. Then, the fitness of each new set is calculated.

Then, the sets are sorted ascendingly according to their fitness value. The best set of solutions is selected, which is the set with the minimum fitness.

F. Selected Event:

At the end, an event must be executed, so one solution must be chosen from the selected set. The nearest event's magnitude of weight for each solution value that satisfies the value of a solution from the best set has been checked, that the absolute difference between it and the event's weight has no negative. The more events found, the more fitness value. The target in the end of Cuckoo Search is to find the set with the minimum fitness. By this way, the chance of selecting the set with previously selected and executed

events is decreased. Without this important step, the statement and branch coverage decrease.

At the first test case, from (1) $N = 0$, so the weights of all the events are equal to 1, and all the sets will have the same fitness. In this case the last event added to the list of extracted events is selected. Then, after selecting an event the bounds change affecting the weights that are used for generating the random solutions. In this case, the fitness of the sets that have previously selected events increases. As the number of selected events at the sets increases, the fitness value increases.

4.2.3 Event Execution

Event Execution module executes the home event if the random value is greater than probability of home event. If not, then it executes the event selected by the previous module at the running application. The selected event is appended to the collection of selected events to be used for the next iteration. The events are prioritized to find the best set of solutions at the end, that has the minimum fitness to prevent redundancy as much as possible between the selected events.

4.2.4 Data Layer

Data layer includes the final output which is a test suite. A test suite includes the associated test cases.

5 Experiments and Results

The aim of the experiments is analysing statement and branch coverage to evaluate the proposed approach compared to monkey, random, frequency and greedy approaches through different applications. We have performed experiments on an emulator of Google Nexus 4 with 5.1 Android version.

5.1 Research Questions

- RQ#1: Is statement coverage maximized when applying the proposed approach compared to monkey, random, frequency and greedy approaches during the same time duration?
- RQ#2: Is branch coverage maximized when applying the proposed approach compared to monkey, random, frequency and greedy approaches during the same time duration?

5.2 Variables Selection

The following subsections mention the used independent and dependent variables in the experiments.

5.2.1 Independent Variables

Independent variables are controlled and modified [Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, 2012]. They are the testing approaches which are monkey

['UI Application Exerciser Monkey - Android Developers', 2020], random, frequency [Machiry et al., 2013], greedy [Adamo et al., 2018b] and the proposed approach.

5.2.2 Dependent Variables

Dependent variables are studied during the experiment to find the impact of changing independent variables [Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, 2012]. They are statement and branch coverage.

5.3 Hypotheses

The following null hypotheses have been proposed to answer the research questions.

- H_{01}: The statement coverage is minimized when applying the proposed approach compared to monkey during the same time duration.
- H_{02}: The statement coverage is minimized when applying the proposed approach compared to random during the same time duration.
- H_{03}: The statement coverage is minimized when applying the proposed approach compared to frequency during the same time duration.
- H_{04}: The statement coverage is minimized when applying the proposed approach compared to greedy during the same time duration.
- H_{05}: The branch coverage is minimized when applying the proposed approach compared to monkey during the same time duration.
- H_{06}: The branch coverage is minimized when applying the proposed approach compared to random during the same time duration.
- H_{07}: The branch coverage is minimized when applying the proposed approach compared to frequency during the same time duration.
- H_{08}: The branch coverage is minimized when applying the proposed approach compared to greedy during the same time duration.

5.4 Experiment Design

The experiments have been performed for 10 test suites with strength of event combination of two events (2-way). Probability of home event of 0.05 was used. Combinatorial-based greedy approach applied in [Adamo et al., 2018b] has been implemented and the results have been compared to its strategy in selecting events. The results have also been compared to monkey ['UI Application Exerciser Monkey - Android Developers', 2020], random [Adamo et al., 2018b] and frequency [Machiry et al., 2013] approaches.

The proposed approach has been experimented on ten applications downloaded from F-Droid which is an open source repository of android applications ['F-Droid: free and open source android app repository', 2020]. They are Who Has My Stuff, Repay, Budget, Loaned, Shopping List, Car Report, Privacy Friendly Notes, Gradr, Budget Watch and A Time Tracker. Who Has My Stuff application is used for tracking

the lent objects by the user. A Time Tracker application is used for tracking the user's activities. Repay is an application for tracking the user's lent money. Budget and Budget Watch are applications used for managing the user's revenue and spending. Loaned is used for tracking the loaned things. Shopping List is used for managing the list of groceries. Car report is used for managing the car's fees. Privacy Friendly Notes is used for managing the user's notes. Gradr is used by the students to track their grades. The details about the version, Uniform Resource Locator (URL), number of statements and branches of each application are represented at Table 2.

They were instrumented by JaCoCo [EclEmma, 2020]. This step enables finding out statement and branch coverage.

For each application, each approach has been run 10 times to get the mean of statement and branch coverage of 10 test suites for each of them. Each experiment has been performed within a testing time duration of 15 minutes. The mean values of dependent variables (statement and branch coverage) were compared. Experiments have been conducted on an emulator of Google Nexus 4 with 5.1 Android version.

Application	**Version**	**Number of Statements**	**Number of Branches**	**URL**
Who Has My Stuff	1.0.25	1026	201	https://f-droid.org/en/packages/de.freewarepoint.whohasmystuff/
A Time Tracker	0.23	1660	533	https://f-droid.org/en/packages/com.markuspage.android.atimetracker/
Repay	1.6	1443	391	https://f-droid.org/en/packages/com.repay.android/
Budget	4.4	2509	694	https://f-droid.org/en/packages/com.notriddle.budget/
Loaned	1.0.2	1956	492	https://f-droid.org/en/packages/com.mattallen.loaned/
Shopping List	0.11.0	1280	344	https://f-droid.org/en/packages/com.woefe.shoppinglist/
Car Report	3.25.0	9021	2870	https://f-droid.org/en/packages/me.kuehle.carreport/
Privacy Friendly Notes	1.0.1	2002	480	https://f-droid.org/en/packages/org.secuso.privacyfriendlynotes/
Gradr	1.1	692	146	https://f-droid.org/en/packages/me.anuraag.grader/
Budget Watch	0.21.5	2122	637	https://f-droid.org/en/packages/protect.budgetwatch/

Table 2: Summary of Mobile Applications Used for Testing

5.5 Analysis and Interpretation

In this section, the experiments' results and hypothesis testing to assess them have been discussed.

5.5.1 Results

The results show that the mean statement and branch coverage of the proposed approach that used metaheuristic Cuckoo search for event selection have been higher than the other approaches.

For Who Has My Stuff application, the proposed approach with combination strength of two events has achieved 71.00% for statement coverage and 45.30% for branch coverage which have been higher than the other approaches. Moreover, the proposed approach has achieved 49.20% for statement coverage and 29.90% for branch coverage at A Time Tracker application which have been higher than the other approaches. For Repay application, the proposed approach has achieved 33.10% for statement coverage and 16.20% for branch coverage. They have been higher compared to the other approaches. Similarly, for Budget, Loaned, Shopping List, Car Report, Privacy Friendly Notes, Gradr and Budget Watch applications, the proposed approach has achieved higher statement and branch coverage compared to the other approaches.
Within the same testing time duration of 15 minutes, Cuckoo 2-way has achieved higher statement and branch coverage results than the other approaches using the ten applications. The mean statement coverage results of the 10 test suites of each of the ten applications within 15 minutes are presented in Table 3. The mean branch coverage results are presented in Table 4. Statement coverage results of the applications have been presented in Figure 2. The statement coverage results presented at the figures show that the proposed approach has achieved the highest statement coverage compared to other approaches. There is huge difference in monkey results, it has achieved the highest statement coverage in two test suites and the lowest results in the rest of test suites as shown in Figure 2 (Car Report). It has also achieved 0% in one test suite as shown in Figure 2 (Budget).

Moreover, Branch coverage results of the applications have been presented in Figure 3. Random has achieved the highest branch coverage results as shown in Figure 3 (Budget), increasing its mean value over the proposed approach. However, the proposed approach has achieved the highest branch coverage for all the other applications as shown in the rest of applications.

Application	**Testing Approach**				
	Monkey ['UI Application Exerciser Monkey - Android Developers', 2020]	Random [Adamo et al., 2018b]	Frequency [Machiry et al., 2013]	Greedy 2-way [Adamo et al., 2018b]	**Proposed Approach**
Who Has My Stuff	15.60%	61.60%	60.50%	63.10%	**71.00%**
A Time Tracker	27.70%	37.40%	40.70%	38.80%	**49.20%**
Repay	13.10%	28.10%	27.20%	28.20%	**33.10%**
Budget	34.60%	43.90%	45.60%	45.10%	**46.40%**
Loaned	11.90%	31.40%	31.70%	32.40%	**33.10%**
Shopping List	45.50%	59.20%	51.80%	64.80%	**66.30%**
Car Report	12.40%	12.60%	16.20%	15.80%	**17.50%**
Privacy Friendly Notes	18.10%	30.60%	30.70%	33.20%	**36.50%**
Gradr	36.00%	46.80%	42.60%	49.00%	**53.20%**
Budget Watch	22.10%	25.50%	28.40%	29.20%	**36.30%**

Table 3: Mean Statement Coverage % of Testing Approaches Using Ten Mobile Applications During the Same Testing Time Duration

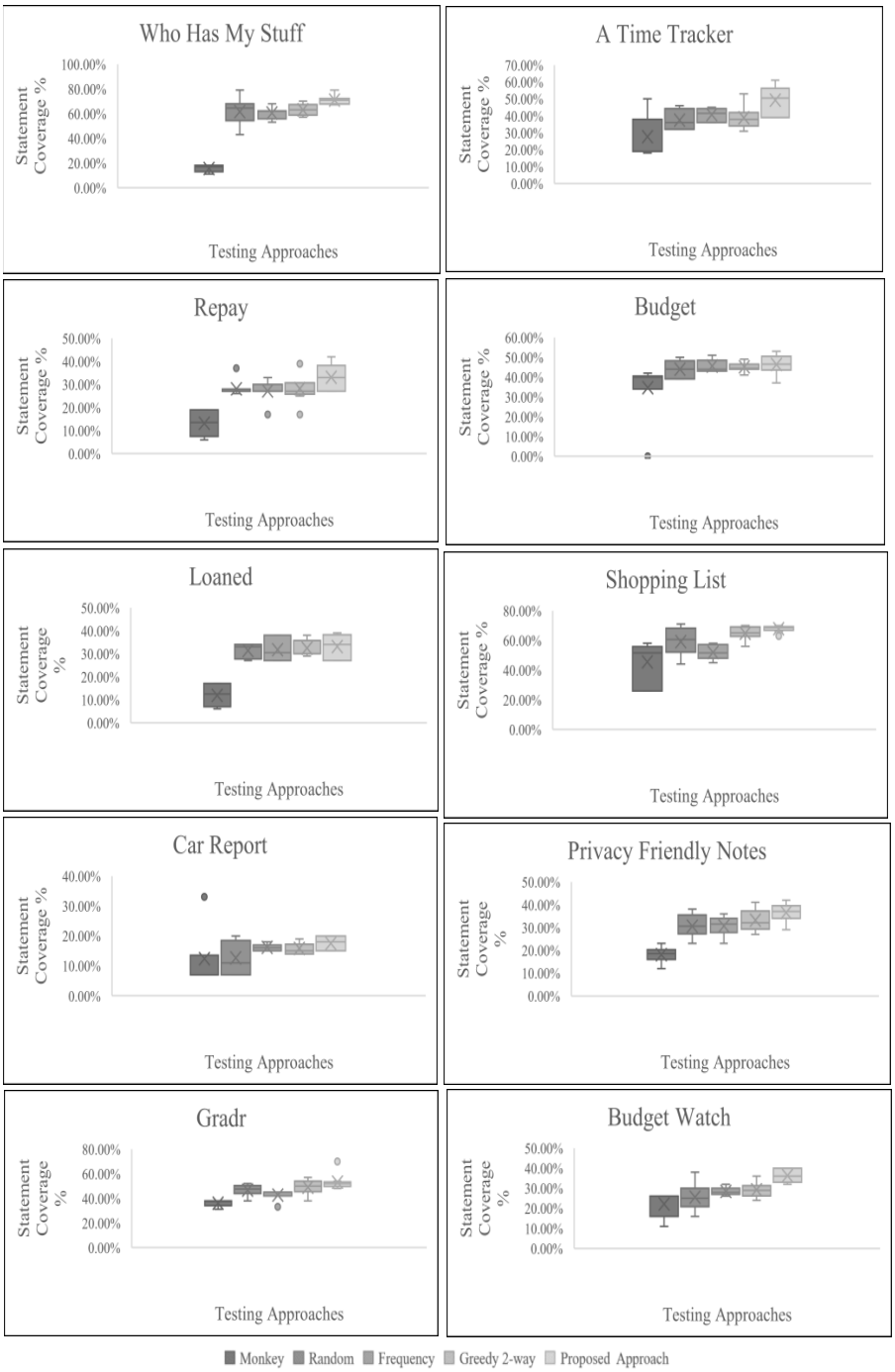

Figure 2: Boxplots visualizing statement coverage % of ten applications

Application	**Testing Approach**				
	Monkey ['UI Application Exerciser Monkey - Android Developers', 2020]	Random [Adamo et al., 2018b]	Frequency [Machiry et al., 2013]	Greedy 2-way [Adamo et al., 2018b]	**Proposed Approach**
Who Has My Stuff	8.10%	36.90%	34.70%	37.80%	**45.30%**
A Time Tracker	21.20%	22.30%	25.20%	23.10%	**29.90%**
Repay	6.90%	14.30%	14.20%	14.30%	**16.20%**
Budget	24.00%	**31.40%**	30.80%	29.40%	30.90%
Loaned	4.70%	15.50%	15.20%	15.00%	**15.70%**
Shopping List	32.00%	37.60%	30.20%	41.70%	**42.00%**
Car Report	7.10%	7.20%	9.50%	9.20%	**10.20%**
Privacy Friendly Notes	10.90%	17.50%	18.20%	19.40%	**21.70%**
Gradr	18.10%	32.10%	30.80%	35.30%	**36.40%**
Budget Watch	10.80%	11.80%	14.60%	13.60%	**17.20%**

Table 4: Mean Branch Coverage % of Testing Approaches Using Ten Mobile Applications During the Same Testing Time Duration

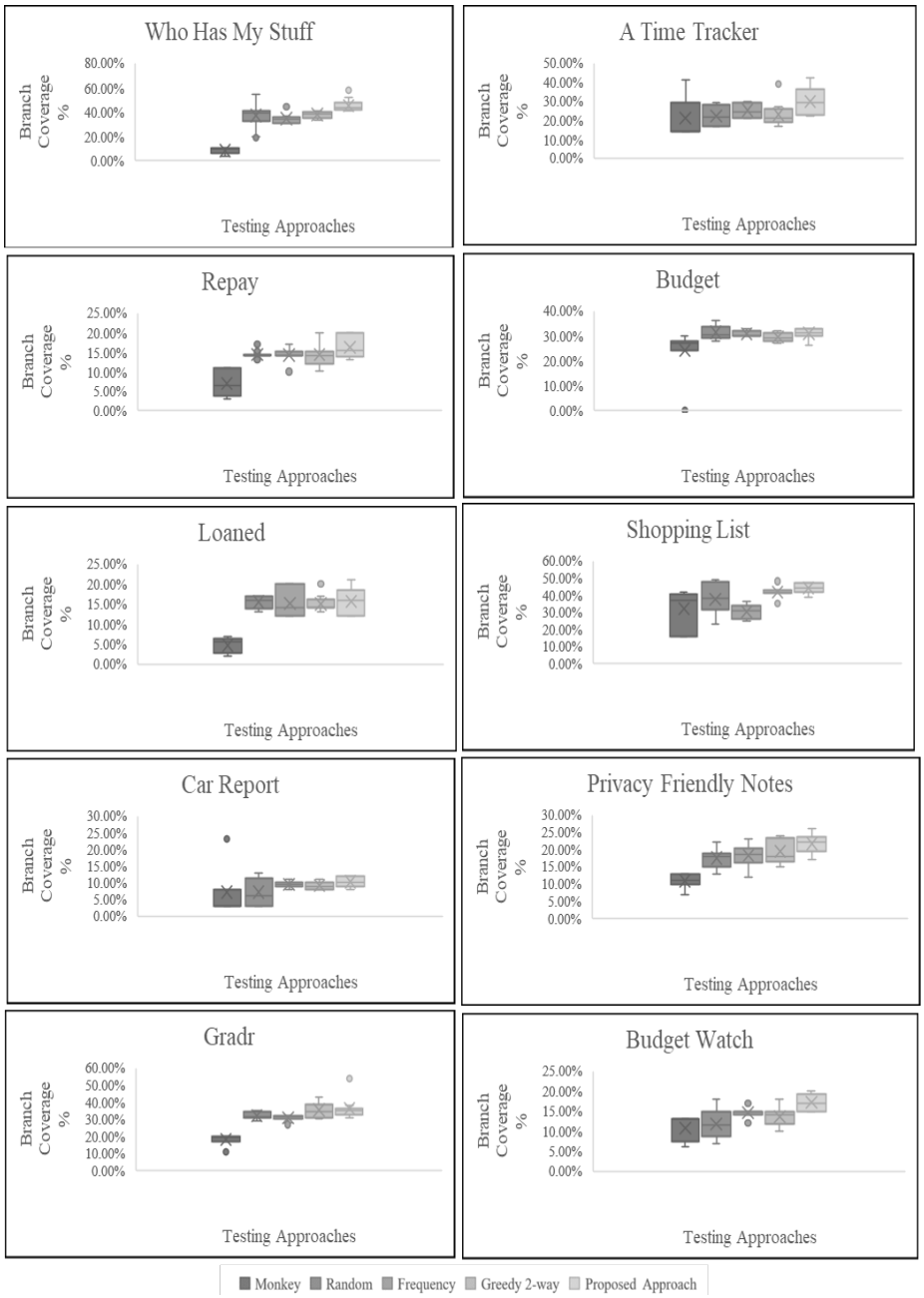

Figure 3: Boxplots visualizing branch coverage % of ten applications

5.5.2 Hypotheses Testing

In this subsection, statistical testing has been applied to address the hypotheses in 5.3. Mann-Whitney test [Mann and Whitney, 1947] has been used. It is a non-parametric test used to compare two groups of variables like the parametric t-test. It is used if there is a likelihood that the data is skewed. All AUTs' values for each approach are merged. *P*-value is calculated for each two approaches, if the *p*-value is less than the threshold for statistical significance (alpha), then the null hypothesis is rejected. Researchers set alpha to 0.05. The goal of the hypothesis testing is to check the effectiveness of the proposed approach, so it is compared to each of the other approaches regarding statement and branch coverages.

Regarding RQ#1 which asks if the statement coverage is maximized when applying the proposed approach compared to monkey, random, frequency and greedy approaches during the same time duration.

The following null hypotheses have been proposed:

- H_{01}: The statement coverage is minimized when applying the proposed approach compared to monkey during the same time duration.
- H_{02}: The statement coverage is minimized when applying the proposed approach compared to random during the same time duration.
- H_{03}: The statement coverage is minimized when applying the proposed approach compared to frequency during the same time duration.
- H_{04}: The statement coverage is minimized when applying the proposed approach compared to greedy during the same time duration.

Mann-Whitney test has been applied to compare between the proposed approach and each of monkey, random, frequency and greedy 2-way where the dependent variable is statement coverage.

P-values are less than alpha (*p*-value < 0.05) as shown in the results at Table 5. Therefore, all the previously mentioned null hypotheses have been rejected.

Thus, the answer of RQ#1 is yes, statement coverage is maximized when applying the proposed approach compared to monkey, random, frequency and greedy approaches during the same time duration.

Hypothesis number	First Approach	Compared Approach	*p*-value
H_{01}	Proposed Approach	Monkey ['UI Application Exerciser Monkey - Android Developers', 2020]	3.39662e-16
H_{02}		Random [Adamo et al., 2018b]	0.001459947
H_{03}		Frequency [Machiry et al., 2013]	0.001489294
H_{04}		Greedy 2-way [Adamo et al., 2018b]	0.018185177

Table 5: P-value Results by Applying Mann-Whitney Test Where Dependent Variable is Statement Coverage

Regarding RQ#2 which asks if the branch coverage is maximized when applying the proposed approach compared to monkey, random, frequency and greedy approaches during the same time duration.
The following null hypotheses have been proposed:

- H_{05}: The branch coverage is minimized when applying the proposed approach compared to monkey during the same time duration.
- H_{06}: The branch coverage is minimized when applying the proposed approach compared to random during the same time duration.
- H_{07}: The branch coverage is minimized when applying the proposed approach compared to frequency during the same time duration.
- H_{08}: The branch coverage is minimized when applying the proposed approach compared to greedy during the same time duration.

Mann-Whitney test has been applied to compare between the proposed approach and each of monkey, random, frequency and greedy 2-way where the dependent variable is branch coverage.

P-values were less than alpha (p-value < 0.05) as shown in the results at Table 6. Therefore, all the previously mentioned null hypotheses have been rejected.
Thus, the answer of RQ#2 is yes, branch coverage is maximized when applying the proposed approach compared to monkey, random, frequency and greedy approaches during the same time duration.

Hypothesis number	First Approach	Compared Approach	*p*-value
H_{05}	Proposed Approach	Monkey ['UI Application Exerciser Monkey - Android Developers', 2020]	7.81824E-13
H_{06}		Random [Adamo et al., 2018b]	0.010573842
H_{07}		Frequency [Machiry et al., 2013]	0.007499698
H_{08}		Greedy 2-way [Adamo et al., 2018b]	0.044280099

Table 6: P-value Results by Applying Mann-Whitney Test Where Dependent Variable is Branch Coverage

5.6 Implications

The results show an increase in statement and branch coverage percentage than monkey, frequency, random and greedy approaches for event selection during the same testing time duration. Monkey has achieved the minimum statement and branch coverage. It generates random events that could be repeated multiple times. They also may not be related to the AUT. Random approach selects any random event from the list of extracted events without a fixed way. It could select the same event multiple times. Frequency approach selects the first event of the list of events that has been selected the minimum number of times. This could be inappropriate when it is required to select the same event multiple times in one GUI state. Greedy approach generates a

candidates list of events by calculating the number of tuples of events that are not covered and based on this value it decides whether to add this event to the candidates list or not. Then, it randomly selects an event from this list to be executed without a fixed way. The proposed approach affects the event selection by using Michalewicz's fitness function (7) and the proposed way of prioritizing and handling repeated events to prevent redundancy in selection, which increase the statement and branch coverage. This increase proves the effectiveness of using Cuckoo search in automated mobile application testing. Besides, it requires Application Under Test Android Application Package (AUT APK) only for testing. Moreover, applying metaheuristic approach is more effective than greedy approach for combinatorial mobile application testing.

5.7 Threats to Validity

In this section threats for the validation of the results of the proposed approach and how they have been solved are discussed. They are threats to internal, construct and external validity.

5.7.1 Threats to Internal Validity

Threats to internal validity are the impacts that could affect the independent variables without the experimenter's interference [Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, 2012, Cook and Campbell, 1979]. Different results could be generated when re-testing an application with the different approaches due to the sequence of the executed events. For this reason, each AUT has been run ten times and ten test suites have been generated. Moreover, comparisons are based on the mean values of statement and branch coverage of the ten test suites assure that there is no bias in the generated results.

5.7.2 Threats to Construct Validity

They are about the problems related to experiment's design to address the concept of the experiment [Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, 2012, Cook and Campbell, 1979]. The concept could not be well clarified if there is one independent variable. However, there are multiple independent variables which are the testing approaches. Moreover, the observations could be biased if there is one metric. Two metrics have been used for assuring the effectiveness of the proposed approach which are statement and branch coverage.

5.7.3 Threats to External Validity

They are the cases that could cause limitations in generalizing the results. Performing the experiments on one time duration of 15 minutes could be a threat. To address that, each approach has been run ten times and ten test suites have been produced. The proposed approach's mean statement and branch coverage results for each application have exceeded the other approaches. That ensures the effectiveness of the proposed approach.

6 Conclusions and Future Work

Nowadays, there are an enormous number of mobile applications launched into the market. Therefore, it is significant to increase the speed of testing using automated approaches to cope with this rapid process. This research proposed an effective automated mobile application GUI testing approach in terms of maximizing statement and branch coverage. The proposed approach is based on a combinatorial metaheuristic Cuckoo search. Experiments were conducted on ten applications within 15 minutes testing time duration and combination strength of two events each in 10 test suites. It has achieved 71.00%, 49.20%, 33.10% for statement coverage and 45.30%, 29.90%, 16.20% for branch coverage of Who Has My Stuff, A Time Tracker and Repay respectively. The proposed approach proved its effectiveness as it prevents redundancy in selection. This leads to having a test suite with a maximized statement and branch coverage compared to monkey, frequency, random and greedy approaches at the same testing time duration. Besides, it requires Application Under Test Android Application Package only for testing. For future work, the approach will be experimented on different testing types other than GUI and on other platforms as testing on Internet of Things (IoT).

References

[Adamo et al., 2018a] Adamo, D., Bryce, R., King, T. M.: 'Randomized event sequence generation strategies for automated testing of android apps'; Information Technology-New Generations, 558 (2018a), 571–578. https://doi.org/10.1007/978-3-319-54978-1_72

[Adamo et al., 2018b] Adamo, D., Nurmuradov, D., Piparia, S., Bryce, R.: 'Combinatorial-based event sequence testing of Android applications'; Information and Software Technology, 99, March (2018b), 98–117. https://doi.org/10.1016/j.infsof.2018.03.007

[Ahmed, 2016] Ahmed, B. S.: 'Test case minimization approach using fault detection and combinatorial optimization techniques for configuration-aware structural testing'; Engineering Science and Technology, an International Journal, 19, 2 (2016), 737–753. https://doi.org/10.1016/j.jestch.2015.11.006

[Alsewari et al., 2020] Alsewari, A. R. A., Poston, R., Zamli, K. Z., Balfaqih, M., Aloufi, K. S.: 'Combinatorial test list generation based on Harmony Search Algorithm'; Journal of Ambient Intelligence and Humanized Computing, , 0123456789 (2020). https://doi.org/10.1007/s12652-020-01696-7

[Appium, 2020] Appium: 'Appium: Mobile App Automation Made Awesome.'; (2020). Retrieved 25 September 2020, from http://appium.io/

[Arnatovich et al., 2018] Arnatovich, Y. L., Wang, L., Ngo, N. M., Soh, C.: 'Mobolic: An automated approach to exercising mobile application GUIs using symbiosis of online testing technique and customated input generation'; Software - Practice and Experience, 48, 5 (2018), 1107–1142. https://doi.org/10.1002/spe.2564

[Artin, 1964] Artin, E.: 'The Gamma Function: Translated by Michael Butler'; HOLT, RINEHART AND WINSTON, New York (1964). Retrieved from http://www.maa.org/publications/maa-reviews/the-gamma-function

['Avocado Testing Framework', 2020] (2020). Retrieved 25 September 2020, from https://avocado-framework.github.io/

[Bajaj and Sangwan, 2021] Bajaj, A., Sangwan, O. P.: 'Discrete cuckoo search algorithms for test case prioritization'; Applied Soft Computing, 110 (2021), 107584. https://doi.org/10.1016/j.asoc.2021.107584

[Bombarda and Gargantini, 2020] Bombarda, A., Gargantini, A.: 'An Automata-Based Generation Method for Combinatorial Sequence Testing of Finite State Machines'; Proceedings - 2020 IEEE 13th International Conference on Software Testing, Verification and Validation Workshops, ICSTW 2020 (2020), 157–166. https://doi.org/10.1109/ICSTW50294.2020.00036

[Bryce et al., 2009] Bryce, R. C., Lei, Y., Kuhn, D. R., Kacker, R.: 'Combinatorial testing'; Handbook of Research on Software Engineering and Productivity Technologies: Implications of Globalization (2009), 196–208. https://doi.org/10.4018/978-1-60566-731-7.ch014

[Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, 2012] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, B. R.: 'Experimentation in Software Engineering'; Springer (Vol. 1) (2012).

[Clement, 2020] Clement, J.: 'Google Play Store: number of apps 2020'; Statista (2020). Retrieved from https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/

[Cook and Campbell, 1979] Cook, T. D., Campbell, D. T.: 'Quasi-experimentation: Design and analysis issues for field settings'; (Vol. 351). Houghton Mifflin Boston (1979).

[Deka et al., 2017] Deka, B., Huang, Z., Franzen, C., Hibschman, J., Afergan, D., Li, Y., et al.: 'Rico: A mobile app dataset for building data-driven design applications'; UIST 2017 - Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology (2017), 845–854. https://doi.org/10.1145/3126594.3126651

[Dustin et al., 2009] Dustin, E., Garrett, T., Gauf, B.: 'Implementing automated software testing: How to save time and lower costs while raising quality'; Pearson Education (2009). Retrieved from http://library1.nida.ac.th/termpaper6/sd/2554/19755.pdf

[EclEmma, 2020] EclEmma: 'JaCoCo Java Code Coverage Library'; (2020). Retrieved 25 September 2020, from http://eclemma.org/jacoco/

['F-Droid: free and open source android app repository', 2020] (2020). Retrieved 25 September 2020, from https://f-droid.org/

[Ferreira and Paiva, 2019] Ferreira, J., Paiva, A. C. R.: 'Android Testing Crawler'; Communications in Computer and Information Science, Vol 1010. Springer, Cham, 2 (2019), 313–326. https://doi.org/10.1007/978-3-030-29238-6_23

[Geem et al., 2001] Geem, Z. W., Kim, J. H., Loganathan, G. V: 'A New Heuristic Optimization Algorithm: Harmony Search'; SIMULATION, 76, 2 (2001), 60–68. https://doi.org/10.1177/003754970107600201

[Hart et al., 1968] Hart, P. E., Nilsson, N. J., Raphael, B.: 'A Formal Basis for the Heuristic Determination of Minimum Cost Paths'; IEEE Transactions of Systems Science and Cybernetics, , 2 (1968), 100–107.

[Homès, 2011] Homès, B.: 'Fundamentals of Software Testing'; Fundamentals of Software Testing. Wiley-ISTE (2011). https://doi.org/10.1002/9781118602270

[Huynh et al., 2019] Huynh, Q. T., Nguyen, D. M., Ha, N. H., Pham, T. K., Nguyen, P. T., Tran, V. D.: 'A combinatorial technique for mobile applications software testing'; Proceedings of 2019 11th International Conference on Knowledge and Systems Engineering, KSE 2019, , October (2019), 1–6. https://doi.org/10.1109/KSE.2019.8919456

[Ince et al., 2014] Ince, D. C., Zhang, J., Zhang, Z., Ma, F.: 'Automatic generation of combinatorial test data'; Springer Berlin Heidelberg (Vol. 30). Springer (2014). https://doi.org/10.1093/comjnl/30.1.63

[Karthikeyani, 2011] Karthikeyani, V.: 'Mobile Software Testing – Automated Test Case Design Strategies'; International Journal, 3, 4 (2011), 1450–1461.

[Kaur, 2015] Kaur, A.: 'Review of Mobile Applications Testing with Automated Techniques'; International Journal of Advanced Research in Computer and Communication Engineering, 4, 10 (2015), 503–507. https://doi.org/10.17148/IJARCCE.2015.410114

[Khan et al., 2018] Khan, R., Amjad, M., Srivastava, A. K.: 'Optimization of automatic test case generation with cuckoo search and genetic algorithm approaches'; Advances in Intelligent Systems and Computing, 554 (2018), 413–423. https://doi.org/10.1007/978-981-10-3773-3_40

[Khari and Kumar, 2017] Khari, M., Kumar, P.: 'An effective meta-heuristic cuckoo search algorithm for test suite optimization'; Informatica (Slovenia), 41, 3 (2017), 363–377.

[Kuhn et al., 2010] Kuhn, D. R., Kacker, R. N., Lei, Y.: 'Practical combinatorial testing'; NIST Special Publication, 800, 142 (2010), 142.

[Kuhn et al., 2013] Kuhn, D. R., Kacker, R. N., Lei, Y.: 'Introduction to combinatorial testing'; CRC press. CRC press (2013). https://doi.org/10.1007/978-3-662-43429-1_1

[Lavrakas, 2008] Lavrakas, D. P. J.: 'Encyclopedia of Survey Research Methods'; The British Journal of Psychiatry (Vol. 112) (2008). https://doi.org/10.1192/bjp.112.483.211-a

[Li et al., 2019] Li, Y., Yang, Z., Guo, Y., Chen, X.: 'Humanoid: A deep learning-based approach to automated black-box android app testing'; Proceedings - 2019 34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019 (2019), 1070–1073. https://doi.org/10.1109/ASE.2019.00104

[Machiry et al., 2013] Machiry, A., Tahiliani, R., Naik, M.: 'Dynodroid : An Input Generation System for Android Apps'; Proceedings of the 9th Joint Meeting on Foundations of Software Engineering, ACM,Pp. 224–234. (2013).

[Mann and Whitney, 1947] Mann, H. B., Whitney, D. R.: 'On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other'; The Annals of Mathematical Statistics, 18, 1 (1947), 50–60. https://doi.org/10.1214/aoms/1177730491

[Mantegna, 1994] Mantegna, R. N.: 'Fast, accurate algorithm for numerical simulation of Lévy stable stochastic processes'; Phys. Rev. E, 49, 5 (1994), 4677–4683. https://doi.org/10.1103/PhysRevE.49.4677

[Memon, 2002] Memon, A. M.: 'SOFTWARE TECHNOLOGIES GUI Testing : Pitfalls and Process'; IEEE Computer (2002), 90–91.

[Michaels et al., 2020] Michaels, R., Adamo, D., Bryce, R.: 'Combinatorial-Based Event Sequences for Reduction of Android Test Suites'; 2020 10th Annual Computing and Communication Workshop and Conference, CCWC 2020 (2020), 598–605. https://doi.org/10.1109/CCWC47524.2020.9031238

[Michalewicz, 1992] Michalewicz, Z.: 'Genetic Algorithms + Data Structures = Evolution Programs'; Springer-Verlag Berlin Heidelberg (Vol. 53). Springer-Verlag Berlin Heidelberg (1992).

[Mirjalili, 2016] Mirjalili, S.: 'SCA: A Sine Cosine Algorithm for solving optimization problems'; Knowledge-Based Systems, 96 (2016), 120–133. https://doi.org/10.1016/j.knosys.2015.12.022

[Molga and Smutnicki, 2005] Molga, M., Smutnicki, C.: 'Test functions for optimization needs';, , c (2005), 1–43.

[Naik and Tripathy, 2011] Naik, K., Tripathy, P.: 'Software testing and quality assurance: theory and practice'; John Wiley & Sons (2011).

[Payne, 2005] Payne, R. B.: 'The Cuckoos'; Bird Families of the World (2005), 618. https://doi.org/10.1017/CBO9781107415324.004

[Richter et al., 2020] Richter, J., Ahmed, B. S., Bures, M., Rosa Junior, C. R., Junior, C. R. R.: 'Avocado: Open-Source Flexible Constrained Interaction Testing for Practical Application'; In 2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW) (2020), 185–190. https://doi.org/10.1109/ICSTW50294.2020.00040

[S. O'Dea, 2020] S. O'Dea: '• Smartphone users 2020 | Statista'; (2020).

[Salihu et al., 2019] Salihu, I. A., Ibrahim, R., Ahmed, B. S., Zamli, K. Z., Usman, A.: 'AMOGA: A Static-Dynamic Model Generation Strategy for Mobile Apps Testing'; IEEE Access, 7 (2019), 17158–17173. https://doi.org/10.1109/ACCESS.2019.2895504

[Samir et al., 2019] Samir, A., Maghawry, H. A., Badr, N.: 'Enhanced Approach for Maximizing Coverage in Automated Mobile Application Testing'; In 2019 Ninth International Conference on Intelligent Computing and Information Systems (ICICIS). IEEE (2019), 402–407. https://doi.org/10.1109/ICICIS46948.2019.9014834

[Sharma et al., 2019] Sharma, S., Rizvi, S. A. M., Sharma, V. K.: 'Test case generation for data flow testing using cuckoo search algorithm'; International Journal of Recent Technology and Engineering, 8, 2 Special Issue 11 (2019), 2953–2964. https://doi.org/10.35940/ijrte.B1377.0982S1119

['UI Application Exerciser Monkey - Android Developers', 2020] (2020). Retrieved 25 September 2020, from https://developer.android.com/studio/test/monkey

[Yang, 2010] Yang, X.-S.: 'Nature Inspired Metaheuristic Algorithms'; LUNIVER PRESS ,UK (2010).

[Yang and Deb, 2009] Yang, X. S., Deb, S.: 'Cuckoo search via Lévy flights'; 2009 World Congress on Nature and Biologically Inspired Computing, NABIC 2009 - Proceedings (2009), 210–214. https://doi.org/10.1109/NABIC.2009.5393690

[Yang and Deb, 2010] Yang, X. S., Deb, S.: 'Engineering optimisation by cuckoo search'; International Journal of Mathematical Modelling and Numerical Optimisation, 1, 4 (2010), 330–343. https://doi.org/10.1504/IJMMNO.2010.035430

[Zamli et al., 2020] Zamli, K. Z., Din, F., Nasser, A. B., Alsewari, A. R.: 'Combinatorial Test Suite Generation Strategy Using Enhanced Sine Cosine Algorithm'; Lecture Notes in Electrical Engineering (Vol. 632). Springer Singapore (2020). https://doi.org/10.1007/978-981-15-2317-5_12

[Zhang et al., 2020] Zhang, L., Yu, Y., Luo, Y., Zhang, S.: 'Improved cuckoo search algorithm and its application to permutation flow shop scheduling problem'; Journal of Algorithms and Computational Technology, 14 (2020). https://doi.org/10.1177/1748302620962403