

Received December 26, 2019, accepted January 10, 2020, date of publication January 13, 2020, date of current version January 22, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2966433

An Empirical Comparison of the Efficiency and Effectiveness of Genetic Algorithms and Adaptive Random Techniques in Data-Flow Testing

FAHAD M. ALMANSOUR¹, ROOBAEA ALROOBAEA^{1,2}, AND AHMED S. GHIDUK^{1,2,3}

¹Department of Computer Science, College of Sciences and Arts in Rass, Qassim University, Buraydah 51452, Saudi Arabia

²College of Computers and Information Technology, Taif University, Taif 21974, Saudi Arabia


³Department of Mathematics and Computer Science, Faculty of Science, Beni-Suef University, Beni-Suef 62521, Egypt

Corresponding author: Ahmed S. Ghiduk (asaghiduk@yahoo.com)

This work was supported by the Deanship of Scientific Research, Qassim University, Saudi Arabia, under Grant alrasscac-2018-1-14-S-3997 during the academic year 1440 AH/2018 AD.

ABSTRACT Software Testing depends on the execution of the tested-program against a set of test-inputs and the comparison of its outputs with the expected ones. The size of the input domain is very large that can be the set of real numbers (\mathbb{R}). Thus, the selection of the appropriate inputs is one of the key problems in software testing. This process is time-consuming and needs a lot of effort and budget. Therefore, automatic inputs generation techniques are required to overcome these problems. Genetic algorithms (GAs) have been successfully used for generating test-inputs. Researchers proved that GAs overcame ordinary random search techniques (ORTs) in generating inputs. In addition, GAs can converge faster than ordinary random techniques and they can reduce effectively the size of the test-suite. Unfortunately, technically GAs needs time more than ORTs. Adaptive random testing technique (ART) is a form of ORT that works for distributing test-cases more evenly through the input domain to increase the efficiency of ORTs. So far, there is no study comparing the efficiency of GAs and ARTs in data-flow testing. In this paper, we introduce an empirical comparison for genetic algorithms and adaptive random techniques according to four factors: reducing the size of the test-suite, convergence speed, elapsed time, and the effectiveness in maximizing the coverage ratio of all du-pairs criterion. The experimental study, which was conducted to compare the two techniques, contains 7 Java programs. The results of the experiments showed that the GA technique defeated the ORT technique and the ART technique in reducing the size of the required test-suite to satisfy all du-pairs criterion. Where the GA technique created in total 31532 test-inputs while the ART technique generated 61841 and the ORT technique produced 32064. Further, the results showed that the GA technique converged faster than the ORT technique and the ART technique. Where the procedure of the GA technique was repeated 3153 times totally while the procedure of the ART technique was iterated 6184 times and the procedure of the ORT technique was repeated 3206 times. In addition, the convergence rate of GA-based technique = 8.25 generations/second, the convergence rate of the ORT = 11.98 generations/second, and the convergence rate of the ART = 13.27 generations/second. Moreover, the results showed that the GA technique is faster than the ART technique and slower than the ORT technique. Where the GA technique consumed in total 382 seconds while the ART technique consumed 465.9 seconds and the ORT technique consumed 267.6 seconds. Additionally, the results showed that the GA technique satisfied overall coverage ratio equals 74% of all du-pairs while the ART technique satisfied 78% and the ORT technique satisfied 73%. From these results, we concluded that GA algorithms are more effective than ORT and ART techniques in data-flow testing but ART satisfied the most coverage ratio. Therefore, we recommend hybridizing GA and ART and applying the hybrid technique in the test-data generation process.

INDEX TERMS Adaptive random testing, data-flow testing, genetic algorithms, test data generation.

The associate editor coordinating the review of this manuscript and approving it for publication was Porfirio Tramontana .

I. INTRODUCTION

Software Testing depends on the execution of the tested-program against a set of test-inputs (test-data) and the

comparison of its outputs to the expected ones [1], [2]. Finding the proper inputs is one of the major difficulties in software testing. This process is a time-consuming process and needs a lot of effort and budget. Therefore, automatic test-inputs generation techniques are required to overcome these problems.

Many test-inputs generation methodologies have been developed [3]. The earliest test-inputs generation technique is the random based inputs generation technique. The random technique relies on producing the inputs randomly until the suitable inputs are found [4]. Symbolic execution based inputs generation: symbolic execution based techniques execute the tested-program using symbolic values instead of actual or numerical values and collect a set of conditions which are solved to find inputs that cover a given target [5] [6]. Dynamic execution based inputs generation techniques depend on the execution of the tested-program using actual or numerical values (e.g., produced randomly), and symbolic values in parallel. Then, path conditions are collected during the execution to find new test-inputs [7], [8]. Search based test-inputs generation techniques have been suggested recently to produce test-inputs. These techniques use techniques such as genetic algorithms [9]–[11], ant colony [12], etc.

Adaptive random testing technique (ART) is a form of ORT that works for distributing test-cases more evenly through the input domain to increase the efficiency of ORTs. A wide number of researches have studied the concept of Adaptive Random Testing (ART) to increase the efficiency of random testing [13]–[16]. ART has been applied in some aspects of software testing such as failure detection [15] and test-data generation for path coverage [16], [17].

Form the above discussion, GAs [37]–[39] and ARTs [40]–[42] are search techniques that have been successfully used in the area of software testing. Researchers demonstrated the efficiency and the effectiveness of GAs which overcame random search in generating test-inputs. In addition, the previous work showed that GA-based test-inputs are effective than those generated randomly. Also, GAs can converge faster than random techniques and they can reduce effectively the size of the test-suite. Unfortunately, technically GAs needs time more than random techniques.

Although several studies have been done to find test-data using genetic algorithms or adaptive random techniques, there is no study compares the efficiency of the two techniques in the area of data-flow testing.

Therefore, this paper provides an empirical study to assess the efficiency and effectiveness of GA and ART in data-flow testing. This study will assess the two techniques according to four factors given below. Therefore, the main contributions of this paper are: 1) developing a testing tool to create the test-inputs using genetic algorithms; 2) developing a testing tool to create the test-inputs using adaptive random technique; 3) conducting an empirical study to examine the following set of research questions:

RQ1: In terms of the number of test-inputs, which one of the two techniques is more effective in reducing the size of the test-suite?

RQ2: In terms of the number of generations, which technique is faster in convergence, GA or ART?

RQ3: In terms of the elapsed time, which technique is faster in time, GA or ART?

RQ4: In terms of the coverage ratio of all du-pairs criterion, which one of the two techniques is more effective in maximizing the coverage ratio, GA or ART?

The main objective of this paper is conducting an empirical comparison for genetic algorithms and adaptive random testing techniques. The comparison of these two techniques will use the following four factors to estimate their efficiency and effectiveness:

i. Reducing the size of the test-suite. The first metric of the comparison is the mass of the test-suite in terms of the number of test-inputs generated by each technique.

ii. Convergence speed. The second metric of the comparison is the convergence in terms of the number of generations needed to find the required test-inputs and the rate of convergence.

iii. Elapsed time. The third metric of the comparison is the execution time of each technique.

iv. Coverage efficiency. The fourth metric of the comparison is the coverage efficiency (i.e. the effectiveness of each technique in data-flow testing especially the coverage of all du-pairs criterion).

A set of experiments will be conducted to compare the two techniques according to these four factors.

The remainder of the paper is organized as follows. Some essential concepts are given in Section 2. Section 3 presents the proposed GAs and ART. Section 4 presents the details of the empirical study and its results are presented in Section 5. Section 6 summaries the results of the empirical study. Section 7 introduces the conclusion of the paper and the upcoming work.

II. BACKGROUND

This section introduces the concepts of test-inputs generation, genetic algorithms, adaptive random techniques, and data-flow testing to facilitate understanding of the rest of this research.

A. TEST-DATA GENERATION

Software testing depends on the execution of the tested component against a number of test cases for detecting defects or assessing quality. A test case is a test element that comprises of: (1) A number of test-inputs (test-suite) which are data items generated by an outside source and delivered to the tested component; (2) Execution conditions needed to run the test such as a specific state of a database; (3) Expected outputs which are the definite results generated by the tested component [1], [2].

Creating test-inputs has a critical role in the success and productivity of the software testing process. This process

concentrates on finding a set of test-inputs to execute the tested-program.

B. GENETIC ALGORITHMS

The main principles of genetic algorithms (GAs) are presented by Holland [18] in 1975. The simple GA begins by creating a preliminary population of individuals. Every one of these individuals is symbolized by a binary string called chromosome randomly created. Figure 1 presents the main steps of the simple GA, in which $P(n)$ is the set of individuals (population) at generation n .

```

Step 1: initialize P(n);
Step 2: evaluate P(n);
Step 3: while termination condition not satisfied do
Step 4:   select P(n+1) from P(n);
Step 5:   recombine P(n+1);
Step 6:   evaluate P(n+1);
         n = n + 1;
end while

```

FIGURE 1. Simple GA algorithm.

In the first step, GA randomly generates a set of individuals as the initial population. In the second step, GA estimates the fitness of every individual in the initial population. In the third step, GA examines whether the termination condition is satisfied or not. In the fourth step, GA picks out couples of individuals to be combined in a selected method creating a new population. In the fifth step, GA mixes the picked individuals by two procedures (crossover and mutation) to create new powerful individuals. The crossover procedure swaps portions of data between two chromosomes while the mutation procedure presents minor changes into a small percentage of the population. Crossover and mutation procedures in GA are important factors to reaching the desired outcomes. As observed in many of GA researches, selecting suitable crossover and mutation testing techniques are objective to the nature of the studied problem, the desired outcomes and data-representation methods [19]. In the sixth step, GA evaluates the fitness of each individual in the current population. These steps are repeated up to either the current population composes a solution to the studied problem or the termination criterion is reached.

C. ADAPTIVE RANDOM TECHNIQUES

A wide number of adaptive random search techniques for solving optimization problems have been presented since 1968 [20]. The first presented Adaptive Random Testing (ART) is the Fixed Size Candidate Set ART algorithm (FSCS-ART) [13] presented in Figure 2. Basically, to select a new test-input, n candidate test-inputs are randomly created. To every candidate c_i , the closest formerly executed test-input is determined, and the distance d_i is calculated. The candidate which has the biggest d_i is chosen, and the others are ignored. This procedure is iterated until the desired stopping condition is reached.

```

T = {} /*T is the set of previously executed test cases*/
randomly generate an input t
test the program using t as test case
add t to T
while (stopping criteria not reached)
  D = 0
  randomly generate next k candidates c1, c2, ..., ck
  for each candidate ci
    calculate the minimum distance di from T
    if di > D
      D = di
      t = ci
    add t to T
  test the program using t as a test case
end while

```

FIGURE 2. FSCS-ART algorithm.

D. DATA-FLOW TESTING

The construction of any program could be represented by the control-flow graph. The control-flow graph $CFG = (V, A)$, distinguished by a single start vertex v_s and a single end vertex v_e , is composed of a group of vertices V , where each vertex represents a single statement or a set of consecutive statements, and a group of directed arcs A , where a directed arc $a = (x, y)$ is an ordered pair of two adjacent vertices, named tail and head of a , respectively [21], [22].

In data-flow testing, all definition-use associations (du-pairs) for a variable z of the tested code are identified. A du-pairs is an ordered triple (d, u, z) in which the statement d holds a definition for the variable z and the statement u holds a use of z that can be reached by d through some paths in the tested code [23], [24]. A variable is defined in a line of code once its value is assigned or changed. A variable is used in a line of code once its value is used in that line and not changed.

There are many data-flow testing criteria such as all uses, all definitions, all du-pairs, etc. [21]–[24]. The aim of data-flow testing is finding a test-suite to cover one of the data-flow testing criteria [21]–[24]. In this work, we use all du-pairs criterion.

III. THE PROPOSED TECHNIQUE

The proposed technique consists of four main phases. In the following sections, these phases will be discussed in more details.

A. ANALYSIS PHASE

This phase analyzes the tested-program and applies the data-flow concepts introduced by Allen and Cocke [25] to find all definition-use pairs (du-pairs) in the tested-program.

B. GENETIC ALGORITHM PHASE

Genetic algorithms have applied in many software testing activates. This phase uses a proposed genetic algorithm to create for the tested-program a suite of test-inputs.

The specifications of the proposed genetic algorithm are described in the following subsections. These specifications are improved from our preceding work [9], [10], and [26] as follows:

1) REPRESENTATION OF THE CHROMOSOME

To represent the solution (the values of the inputs of the tested-program), a binary array of length l which consists of a set of zeros and ones (e.g., $c = 10101001$, $l = 8$ for example for two integers input variable) is used by the suggested genetic algorithm to encode the set of inputs of the tested-program. The length (l) of this array (i.e., number of zeros and ones) is subject to the demanded precision and the data type of each variable. This binary array is called chromosome which can be decoded to represent a solution of the studied problem or a set of test-inputs for the tested-program (e.g. chromosome c can be decoded to test input 10 and 9).

2) INITIAL POPULATION

Each population (test-suite T) is consists of ps of binary arrays which are called chromosomes. Each chromosome is a test input t . The first population is called the initial population. Consistent with the representation of chromosome, the initial population is a set of size ps of binary arrays or chromosomes which are built randomly by creating ps bit arrays of length l . The accepted size of the population is experimentally determined. Each chromosome can be decoded into a number of values represent k test input variables or a test input.

3) EVALUATION FUNCTION

The suggested genetic algorithm employs an evaluation function to estimate the quality of the created inputs. This function aims at maximizing the coverage ratio of the du-pairs in the tested-program. The function is written as a formula in terms of the number of covered du-pairs and the total number of du-pairs. This formula is encoded as given in eq. (1).

$$F = \frac{\text{number of covered du_pairs}}{\text{Total number of du_pairs}} \quad (1)$$

4) SELECTION

The suggested GA utilizes the roulette wheel technique [19] to select from the current population of test-inputs a set of inputs (parent individuals) to be recombined by two GA-operations (GA-crossover and GA-mutation) for constructing ps new inputs as the new population.

5) RECOMBINATION

The recombination procedure applies GA-crossover and GA-mutation processes on the data selected by the roulette wheel technique to construct new individuals to form a new population.

GA-Crossover creates more quality individuals over time by swapping chunks of data at a random position between couples of individuals in the current population. The subset

of individuals which subject to the crossover process depends on crossover probability px and population size ps . The size of this subset is $px \times ps$. The most suitable values of ps and px depends on the experiment.

GA-Mutation changes the values of some cells inside a selected individual from 0 to 1 or vice versa. The number of mutated cells of an individual depends on a pre-determined probability pm , length of the individual l , and the size of population ps . The number of mutated cells can be computed using the formula $pm \times l \times ps$.

6) GA-STOP CONDITION

In each cycle of the suggested genetic algorithm, the population is evolved until creating test-inputs that cover the target du-pairs or the maximum number of generations $maxGen$ is reached. In addition, the suggested genetic algorithm is reiterated until covering a percentage of all du-pairs.

C. ADAPTIVE RANDOM PHASE

This module implements the concepts of adaptive random testing techniques [13]–[15], [20] to generate a set of test-inputs for each tested-program. The first ART technique, (FSCS-ART) [13], is given in Figure 2. The specifications of the proposed adaptive random technique are described in the following subsections. The major components of the proposed ART are adapted from the work of T. Y. Chen [13]–[15] as follows:

1) INITIAL TEST INPUT

The ART starts by generating randomly only one test input t as the initial population. The ART executes the tested-program against the test input t . Then ART keeps the test input t in the test-suite T .

2) NEW TEST INPUTS

ART creates randomly k candidates test-inputs c_i and $i = 1 \dots k$.

3) EVALUATION AND SELECTION

To select the new test input, ART locates for each candidate test-input c_i the closest previously executed test-input and determines the distance d_i . Then, ART selects the candidate with the largest d_i and discards the others.

4) ART-STOP CONDITION

This process is stopped once a percentage of all du-pairs are covered by a set of test-inputs or the maximum number of generation ($mGen$) is reached.

D. DATA-FLOW TESTING PHASE

This module aims at estimating the quality of each one of the two test-suites which are generated in the earlier phases by the proposed genetic algorithm and the suggested adaptive random technique. To achieve its goal, the data-flow testing module executes each tested-program against each one of

the two suites. Then for each tested-program, the size of the test-suite, the number of generations, the elapsed time and the coverage ratio of all du-pairs for each technique are calculated.

Any data-flow testing criterion can be used instead of all du-pairs criterion [22], [24]. Besides, mutation testing can be used to evaluate the quality of the test-suites [27].

IV. THE PROPOSED EMPIRICAL STUDY

In this section, the main stages of the empirical study such as the implemented prototype, the subject programs, the conducted experiments to answer the proposed research questions and the setup and procedure of the experiments are discussed. While the results of these experiments will be discussed in the next section.

A. PROTOTYPE IMPLEMENTATION

To answer the research questions, a set of empirical studies was conducted to investigate these questions. Therefore, an automatic tool was developed which consists of four modules: Analysis-Module, GA-Module, ART-Module, and Data-flow-Testing Module. Figure 3 shows the architecture of the prototype tool.

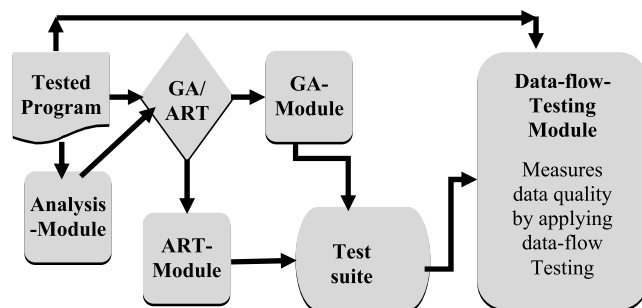


FIGURE 3. Architecture of the prototype tool.

The tool starts by reading the program under test in Java language. Then, the tool calls the analysis module to find the set of all du-pairs. After that, it passes the tested-program and the set of all du-pairs to the GA-Module and ART-Module. Then, it selects one of two modules (GA-Module or ART-Module) to generate a test-suite to cover a selected data-flow criterion such as all-du-pairs. Finally, the tool passes the generated test-suites to the Data-flow Testing Module which executes each tested-program using the generated test-suite. Then, the data-flow testing module estimates the quality of the test-suite by calculating for each tested-program the size of the test-suite, the number of generations, the elapsed time and the coverage ratio of all du-pairs.

B. SUBJECT PROGRAMS

The conducted experiment used a set of subject Java programs that have been used in previous similar researches [9], [28]–[32]. These subjects contain a set of benchmarks programs such as Mid, Remainder, Triangle, and Power, and

some artificial programs of various configurations and structures. Table 1 gives the specifications of these programs: the first column presents code and name for each program; the second column gives several of the previous work which employed these programs; the third column presents the specifications of each program. Each program is treated distinctly from the other programs. Consequently, the procedure of the experiment is conducted separately on each program and each research question is investigated separately.

TABLE 1. The specifications subjects.

# Tested-program	Reference	#LOC; #Classes; #Methods
P#1. Triangle	[28, 29, 30, 9, 31, 32]	73; 1; 6
P#2. Mid	[30, 9, 33, 31, 32]	61; 1; 6
P#3. Power	[9, 34, 33, 31, 32]	49; 1; 5
P#4. Remainder	[9, 34, 33, 31, 32]	60; 1; 5
P#5. Synthetic1	[9, 31, 32]	65; 1; 5
P#6. Synthetic2	[9, 31, 32]	60; 1; 5
P#7. Synthetic3	[9, 31, 32]	62; 1; 5

C. THE SETUP AND PROCEDURE OF THE EMPIRICAL STUDY

Setup the specification of the proposed genetic algorithm to be: $ps = 10, 50, \text{ and } 100$; $px = 0.85$; $pm = 0.15$; $maxGen = 100$. In addition and to fairness, setup the specification of the proposed adaptive random technique to be: the number of candidates test-inputs $k = 10$ as recommended by Chen et al. [13]; $mGen = 100$.

For each tested-program apply the following steps ten times with changing ps to be 10, 50, or 100:

- 1- Apply the analysis module on the tested-program to find its du-pairs.
- 2- For each du-pair, do the following steps:
 - a- Select ps test-inputs and apply the proposed GA to find a test-suite (GA-T) that covers the selected du-pair.
 - b- Select k test-inputs and apply the proposed ART to find a test-suite (ART-T) that covers the selected du-pair.
 - c- Pass the tested-program, the selected du-pair and the two test-suite GA-T and ART-T to the data-flow testing module which executes the tested-program against each one of the two test-suites.
 - d- For each tested-program, record the following four metrics: the size of the test-suites GA-T and ART-T, the number of generations needed to create each test-suite, elapsed time, and coverage ratio of all du-pairs.
 - e- Go to step (a).
- 3- find the average of the ten times for each metric.
- 4- Go to step (1).

V. RESULTS AND DISCUSSION

To answer the research questions, the procedure of the experiment is performed ten times such that in the first three

TABLE 2. No. of test-data generated by GA.

Tested-program	Run number										min	Max	Average
	Run1	Run2	Run3	Run4	Run5	Run6	Run7	Run8	Run9	Run10			
P#1	120	50	1050	170	1010	1120	3010	30	1010	10010	30	10010	1758
P#2	40	7010	7010	4010	5010	6010	7010	4010	190	7010	40	7010	4731
P#3	6000	6000	1010	2020	2040	5000	1020	1010	6000	6000	1010	6000	3610
P#4	12010	12010	12010	12010	6010	6010	6010	6010	10010	6660	6010	12010	8875
P#5	4010	4010	4010	4510	4040	6010	6010	4010	6010	4370	4010	6010	4699
P#6	1030	2500	1560	1980	2580	3450	1000	3910	4000	2470	1000	4000	2448
P#7	4010	6000	5010	5030	5010	4010	5010	5010	4010	11010	4010	11010	5411
Total											16110	56050	31532
Average											2301.4	8007.1	4504.6

TABLE 3. No. of test-data generated by ORT.

Tested-program	Run number										min	Max	average
	Run1	Run2	Run3	Run4	Run5	Run6	Run7	Run8	Run9	Run10			
P#1	10	260	1030	300	1030	1020	3010	950	1020	11010	10	11010	1964
P#2	30	7010	7010	4010	7010	7010	7010	4010	90	7010	30	7010	5020
P#3	6000	6000	2010	2030	4000	5000	1020	1010	6000	6000	1010	6000	3907
P#4	12010	12010	12010	12010	7010	6850	6030	8010	810	8120	810	12010	8487
P#5	4010	4040	4010	4120	4020	6010	6010	4010	6010	6010	4010	6010	4825
P#6	1030	2500	1560	1980	2580	3450	1000	3910	4000	2470	1000	4000	2448
P#7	4010	6000	5030	5030	5010	4010	5010	5010	4010	11010	4010	11010	5413
Total											10880	57050	32064
Average											1554.3	8150.0	4580.6

TABLE 4. No. of test-data generated by ART.

Tested-program	Run number										Min	Max	Average
	Run1	Run2	Run3	Run4	Run5	Run6	Run7	Run8	Run9	Run10			
P#1	20	270	2080	330	2050	2040	6030	970	2030	22020	20	22020	3784
P#2	70	14020	14020	8020	14020	14020	14020	8020	140	14020	70	14020	10037
P#3	12000	12000	3020	4080	3040	10000	2480	2020	12000	12000	2020	12000	7264
P#4	24020	24020	24020	24020	13020	12860	12510	14020	15020	14860	12510	24020	17837
P#5	8020	8060	8020	8230	8040	12020	12020	8030	12020	12020	8020	12020	9648
P#6	1030	2500	1560	1980	2580	3450	1000	3910	4000	2470	1000	4000	2448
P#7	8020	12000	10050	10040	10020	8020	10020	10020	8020	22020	8020	22020	10823
Total											31660	110100	61841
Average											4522.9	15728.6	8834.4

times ps is configured to be 10; in the second three times ps is configured to be 50; in the last four times ps is configured to be 100. Besides, an ordinary random testing technique (ORT) is implemented to be a baseline and compared with GA and ART. The number of the generated test-inputs, the number of generations, the elapsed time, and the coverage ratio of all du-pairs for each subject program are computed. Finally, the total and the average of all ten runs are computed for the number of the generated test-inputs, the number of generations, the elapsed time, and the coverage ratio.

Exploring the first research question (RQ1: In terms of the number of test-inputs, which one of the two techniques is more effective in reducing the size of the test-suite?):

After applying the procedure of the empirical study as given in section IV.C, the number of test-inputs was gathered and presented as follows. Table 2 presents the number of test-inputs generated by the GA technique. Table 3 presents the number of test-inputs generated by the ORT technique. Table 4 presents the number of test-inputs generated by the ART technique.

Figure 4 gives a comparison between the GA, the ORT, and the ART techniques according to the number of test-inputs generated by each of them for every subject program. Figure 5 gives a comparison between the GA, the ORT, and the ART techniques according to the total number of test-inputs generated by each of them for all subject programs and the average as well.

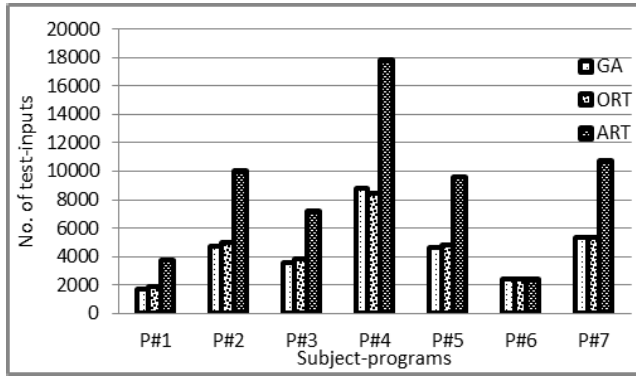


FIGURE 4. No. of generated test-inputs for each program using the GA, the ORT, and the ART techniques.

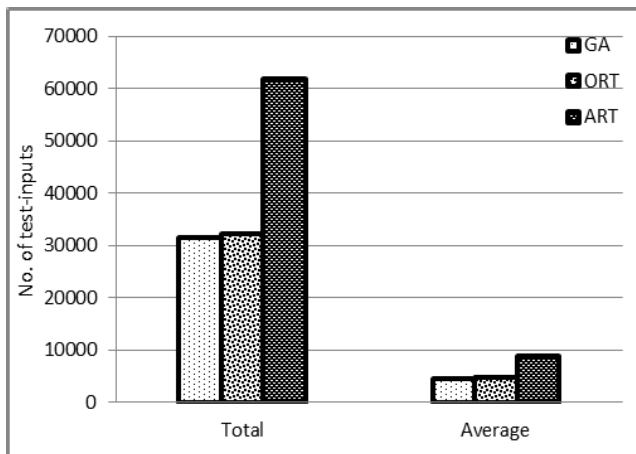


FIGURE 5. The total and average of no. of test-inputs generated by the GA, the ORT, and the ART techniques.

According to the results given in Table 2, Table 3, Table 4, Figure 4, and Figure 5, the GA technique is the most effective technique in reducing the size of the test-suite. The GA technique creates in total 31532 test-inputs to cover 74% of all du-pairs of all subject programs with an average 4504.6 test-inputs for each subject program while the ART technique generates in total 61841 test-inputs to cover 78% of all du-pairs of all subject programs with an average 8834.4 test-inputs for each subject program. In addition, the ORT technique generates in total 32064 test-inputs to cover 73% of all du-pairs of all subject programs with an average 4580.6 test-inputs for each subject program. Besides, the GA technique generates for each subject program a test-suite smaller than the test-suite generated by the ORT technique and the ART technique. Consequently, the GA technique defeated the ORT technique and the ART technique in reducing the size of the required test-suite to cover the du-pairs of each subject programs.

Exploring the second research question (RQ2: In terms of the number of generations, which technique is faster, the GA or the ART?):

After applying the procedure of the empirical study as given in section IV.C, the number of generations was gathered

and presented as follows. Table 5 presents the number of generations taken by the GA technique to create a test-suite for covering the du-pairs of the subject programs. In addition, Table 6 presents the number of generations taken by the ORT technique to find that test-suite. Besides, Table 7 presents the number of generations taken by the ART technique to find that test-suite. Figure 6 gives a comparison between the GA, the ORT and the ART techniques according to the number of generations taken by each of them for every subject program. Figure 7 gives a comparison between the GA, the ORT and the ART according to the total number of generations taken by each of them for all subject programs and the average value of number of generations as well.

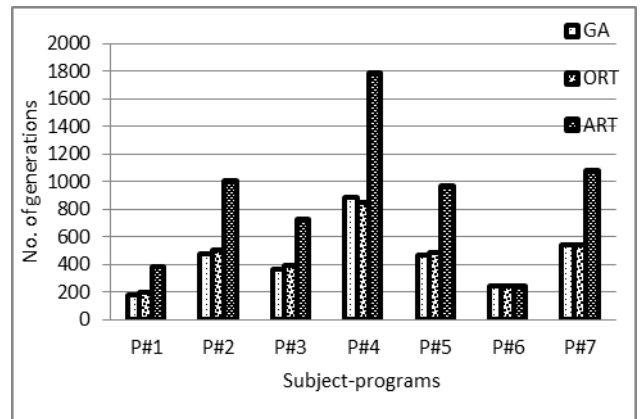


FIGURE 6. No. of generations taken by the GA, the ORT and the ART techniques.

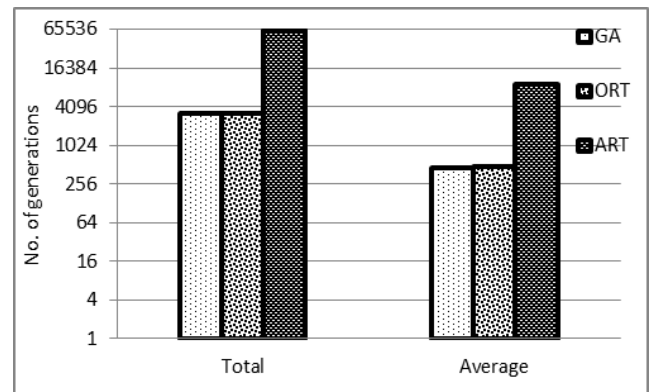


FIGURE 7. Total and average of no. of generations taken by the GA, the ORT, and the ART techniques.

According to the results given in Table 5, Table 6, Table 7, Figure 6 and Figure 7 the GA technique converges faster than the ORT technique and the ART technique. Where the procedure of GA was repeated in total 3153 times to cover 74% of all du-pairs of all subject programs with an average 450.5 times for each subject program while the procedure of the ART technique was iterated in total 6184 times to cover 78% of all du-pairs of all subject programs with an average 883.4 times for each subject program. In addition, the procedure of the ORT technique was repeated in total 3206 times

TABLE 5. No. of generation taken by GA.

Tested-program	Run number										Min	Max	average
	Run1	Run2	Run3	Run4	Run5	Run6	Run7	Run8	Run9	Run10			
P#1	12	5	105	17	101	112	301	3	101	1001	3	1001	175.8
P#2	4	701	701	401	501	601	701	401	19	701	4	701	473.1
P#3	600	600	101	202	204	500	102	101	600	600	101	600	361
P#4	1201	1201	1201	1201	601	601	601	601	1001	666	601	1201	887.5
P#5	401	401	401	451	404	601	601	401	601	437	401	601	469.9
P#6	103	250	156	198	258	345	100	391	400	247	100	400	244.8
P#7	401	600	501	503	501	401	501	501	401	1101	401	1101	541.1
Total											1611	5605	3153.2
Average											230.1	800.7	450.5
Convergence rate = number of generations / elapsed time													8.25

TABLE 6. No. of generation taken by ORT.

Tested-program	Run number										Min	Max	average
	Run1	Run2	Run3	Run4	Run5	Run6	Run7	Run8	Run9	Run10			
P#1	1	26	103	30	103	102	301	95	102	1101	1	1101	196.4
P#2	3	701	701	401	701	701	701	401	9	701	3	701	502
P#3	600	600	201	203	400	500	102	101	600	600	101	600	390.7
P#4	1201	1201	1201	1201	701	685	603	801	81	812	81	1201	848.7
P#5	401	404	401	412	402	601	601	401	601	601	401	601	482.5
P#6	103	250	156	198	258	345	100	391	400	247	100	400	244.8
P#7	401	600	503	503	501	401	501	501	401	1101	401	1101	541.3
Total											1088	5705	3206.4
Average											155.4	815.0	458.1
Convergence rate = number of generations / elapsed time													11.98

TABLE 7. No. of generation taken by ART.

Tested-program	Run number										min	Max	average
	Run1	Run2	Run3	Run4	Run5	Run6	Run7	Run8	Run9	Run10			
P#1	2	27	208	33	205	204	603	97	203	2202	2	2202	378.4
P#2	7	1402	1402	802	1402	1402	1402	802	14	1402	7	1402	1003.7
P#3	1200	1200	302	408	304	1000	248	202	1200	1200	202	1200	726.4
P#4	2402	2402	2402	2402	1302	1286	1251	1402	1502	1486	1251	2402	1783.7
P#5	802	806	802	823	804	1202	1202	803	1202	1202	802	1202	964.8
P#6	103	250	156	198	258	345	100	391	400	247	100	400	244.8
P#7	802	1200	1005	1004	1002	802	1002	1002	802	2202	802	2202	1082.3
Total											3166	11010	6184.1
Average											452.3	1572.9	883.4
Convergence rate = number of generations / elapsed time													13.27

to cover 73% of all du-pairs of all subject programs with an average 458.1 times for each subject program. Besides, the GA technique procedure was iterated for each subject program number of generations smaller than the procedures

of the ORT technique and the ART technique. Consequently, the GA technique overcomes the ORT technique and the ART technique in reducing the number of generations to cover the du-pairs of each subject programs.

TABLE 8. Elapsed time taken by GA.

Tested-program	Run number										min	Max	Average
	Run1	Run2	Run3	Run4	Run5	Run6	Run7	Run8	Run9	Run10			
P#1	1	1	2	1	2	3	16	1	2	173	1	173	20.2
P#2	1	183	96	51	67	70	173	97	1	187	1	187	92.6
P#3	80	68	3	16	31	54	5	6	62	110	3	110	43.5
P#4	154	178	159	133	43	42	44	43	114	54	42	178	96.4
P#5	18	25	24	32	47	62	62	24	50	20	18	62	36.4
P#6	23	30	40	25	100	125	60	75	45	69	23	125	59.2
P#7	18	39	27	34	36	17	30	9	22	105	9	105	33.7
Total											97	940	382
Average											13.9	134.3	54.6

TABLE 9. Elapsed time taken by ORT.

Tested-program	Run number										min	Max	Average
	Run1	Run2	Run3	Run4	Run5	Run6	Run7	Run8	Run9	Run10			
P#1	1	1	2	1	3	2	15	2	2	204	1	204	23.3
P#2	1	185	98	60	126	87	175	83	1	184	1	185	100
P#3	15	11	5	15	20	21	3	5	18	42	3	42	15.5
P#4	22	27	31	19	18	20	18	33	40	50	18	50	27.8
P#5	4	26	20	18	44	47	124	19	36	7	4	124	34.5
P#6	23	30	40	25	100	125	60	75	45	69	23	125	59.2
P#7	3	7	6	8	7	3	9	7	7	16	3	16	7.3
Total											53	746	267.6
Average											7.6	106.6	38.2

To find the convergence speed, we computed for each technique of the three techniques (GA, ORT, and ART) the convergence rate using the following formula:

$$\text{convergence rate} = \frac{\text{number of generations}}{\text{elapsed time}}$$

We found that the convergence rate of GA-based technique = 8.25 generations/second, the convergence rate of the ORT = 11.98 generations/second, and the convergence rate of the ART = 13.27 generations/second. Consequently, the GA-based technique converged faster than the ORT technique and the ART technique.

Exploring the third research question (RQ3: In terms of the elapsed time, which technique is faster, the GA technique or the ART technique?):

After applying the procedure of the empirical study as given in section IV.C, the elapsed time was gathered and presented as follows. Table 8 presents the elapsed time taken by the GA technique to create a test-suite for covering the du-pairs of the subject programs. In addition, Table 9 presents the elapsed time taken by the ORT technique to find that test-suite. Besides, Table 10 presents the elapsed time taken by the ART technique to find that test-suite. Figure 8 gives a

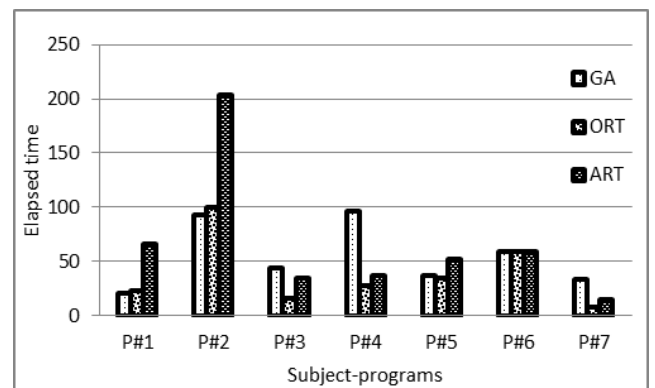


FIGURE 8. Elapsed time taken by the GA, the ORT and the ART techniques for each subject program.

comparison between the GA technique, the ORT technique and the ART technique according to the elapsed time taken by each of them for every subject program. Figure 9 gives a comparison between the GA technique, the ORT technique and the ART technique according to the total elapsed time taken by each of them for all subject programs and the average value of the elapsed time as well.

TABLE 10. Elapsed time taken by ART.

Tested-program	Run number										min	Max	Average
	Run1	Run2	Run3	Run4	Run5	Run6	Run7	Run8	Run9	Run10			
P#1	1	1	6	1	9	7	30	1	9	591	1	591	65.6
P#2	1	309	253	202	337	223	294	129	1	289	1	337	203.8
P#3	45	18	6	45	77	45	20	22	27	42	6	77	34.7
P#4	31	38	41	26	21	22	29	33	52	71	21	71	36.4
P#5	6	60	39	26	110	87	70	39	66	11	6	110	51.4
P#6	23	30	40	25	100	125	60	75	45	69	23	125	59.2
P#7	6	14	11	14	15	6	12	16	10	44	6	44	14.8
Total											64	1355	465.9
Average											9.1	193.6	66.6

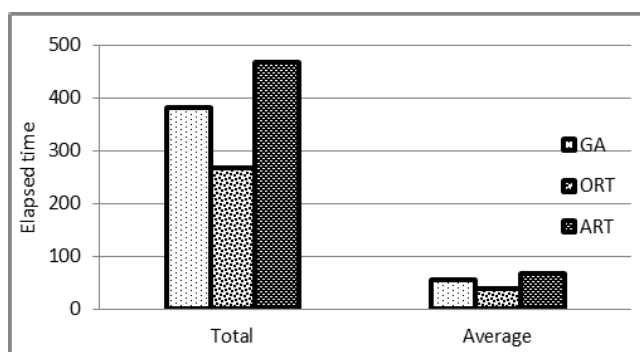


FIGURE 9. Total and average of elapsed time taken by the GA, the ORT and the ART techniques.

According to the results given in Table 8, Table 9, Table 10, Figure 8 and Figure 9 the GA technique is faster than the ART technique and slower than the ORT technique. Where the GA technique consumed in total 382 seconds to cover 74% of all du-pairs of all subject programs with an average 54.6 seconds for each subject program while the ART technique consumed in total 465.9 seconds to cover 78% of all du-pairs of all subject programs with an average 66.6 seconds for each subject program. In addition, the ORT technique consumed in total 267.6 seconds to cover 73% of all du-pairs of all subject programs with an average 38.2 seconds for each subject program. Besides, the GA technique consumed time smaller than the ART technique for three subject programs out of seven programs while the ART technique consumed time smaller than the GA technique for other three subject programs. While the ORT technique consumed time smaller than the GA technique and the ART technique for all subject programs. These results due to the nature of each technique where the processes of the GA procedure and the processes of the ART procedure are more than the processes of the ORT procedure.

Exploring the fourth research question (RQ4: In terms of the coverage ratio of all du-pairs criterion, which one of the two techniques is more effective in generating a high quality test-suite, the GA technique or the ART technique?):

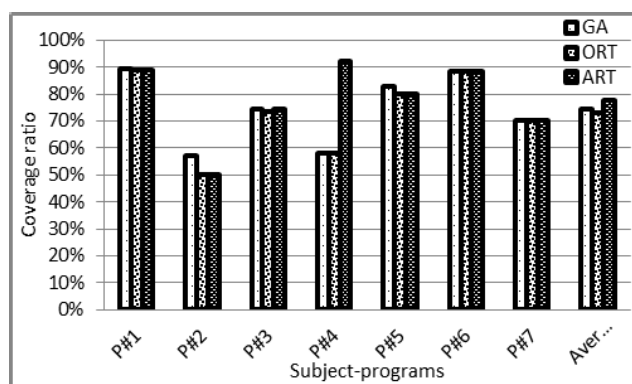


FIGURE 10. Coverage ratio done by the GA, the ORT and the ART techniques for each subject program and its average.

After applying the procedure of the empirical study as given in section IV.C, the coverage ratio of du-pairs was gathered and presented as follows. Table 11 presents the coverage ratio of du-pairs done by the test-suite generated using the GA technique for each subject program. In addition, Table 12 presents the coverage ratio of du-pairs done by the test-suite generated using the ORT technique for each subject program. Besides, Table 13 presents the coverage ratio of du-pairs done by the test-suite generated using the ART technique for each subject program. Figure 10 gives a comparison between the GA technique, the ORT technique and the ART technique according to the coverage ratio of du-pairs done by each of them for every subject program.

According to the results given in Table 11, Table 12, Table 13 and Figure 10 the coverage ratio of du-pairs done by the ART technique is higher than the coverage ratio of du-pairs done by the GA technique and the ORT technique. Where the GA technique satisfied coverage ratio equals 74% of all du-pairs for all subject programs while the ART technique satisfied coverage ratio equals 78% of all du-pairs for all subject programs. In addition, the ORT technique satisfied coverage ratio equals 73% of all du-pairs for all subject programs.

TABLE 11. The coverage ratio of du-pairs done by GA.

Tested-program	Run number										min	Max	Average
	Run1	Run2	Run3	Run4	Run5	Run6	Run7	Run8	Run9	Run10			
P#1	100%	100%	94%	100%	94%	94%	81%	100%	94%	38%	38%	100%	89%
P#2	100%	50%	30%	60%	50%	60%	30%	60%	100%	30%	30%	100%	57%
P#3	57.1%	57.1%	92.9%	85.7%	85.7%	64.3%	92.9%	92.9%	57.1%	57.1%	57.1%	92.9%	74.3%
P#4	40%	40%	40%	40%	70%	70%	70%	70%	70%	70%	40%	70%	58%
P#5	83.3%	83.3%	83.3%	83.3%	83.3%	75.0%	83.3%	83.3%	83.3%	83.3%	75.0%	83.3%	82.5%
P#6	80.0%	95.0%	95.0%	100.0%	80.0%	80.0%	80.0%	90.0%	90.0%	95.0%	80.0%	100.0%	88.5%
P#7	77.8%	66.7%	72.2%	72.2%	72.2%	77.8%	72.2%	72.2%	77.8%	38.9%	38.9%	77.8%	70.0%
Average											51%	89%	74%

TABLE 12. The coverage ratio of du-pairs done by ORT.

Tested-program	Run number										min	Max	Average
	Run1	Run2	Run3	Run4	Run5	Run6	Run7	Run8	Run9	Run10			
P#1	100%	100%	94%	100%	94%	94%	81%	100%	94%	31%	31%	100%	89%
P#2	100%	30%	30%	60%	30%	30%	30%	60%	100%	30%	30%	100%	50%
P#3	57.1%	57.1%	85.7%	85.7%	85.7%	64.3%	92.9%	92.9%	57.1%	57.1%	57.1%	92.9%	73.6%
P#4	40%	40%	40%	40%	70%	70%	70%	70%	70%	70%	40%	70%	58%
P#5	83.3%	83.3%	83.3%	83.3%	83.3%	75.0%	75.0%	83.3%	75.0%	75.0%	75.0%	83.3%	80.0%
P#6	80.0%	95.0%	95.0%	100.0%	80.0%	80.0%	80.0%	90.0%	90.0%	95.0%	80.0%	100.0%	88.5%
P#7	77.8%	66.7%	72.2%	72.2%	72.2%	77.8%	72.2%	72.2%	77.8%	38.9%	38.9%	77.8%	70.0%
Average											50%	89%	73%

TABLE 13. The coverage ratio of du-pairs done by ART.

Tested-program	Run number										min	Max	Average
	Run1	Run2	Run3	Run4	Run5	Run6	Run7	Run8	Run9	Run10			
P#1	100%	100%	93.8%	100%	93.8%	93.8%	81.3%	100%	93.8%	31.3%	31%	100%	89%
P#2	100%	30%	30%	60%	30%	30%	30%	60%	100%	30%	30%	100%	50%
P#3	57.1%	57.1%	92.9%	85.7%	85.7%	64.3%	92.9%	92.9%	57.1%	57.1%	57.1%	92.9%	74.3%
P#4	80%	80%	80%	80%	100%	100%	100%	100%	100%	100%	80%	100%	92%
P#5	83.3%	83.3%	83.3%	83.3%	83.3%	75.0%	75.0%	83.3%	75.0%	75.0%	75.0%	83.3%	80.0%
P#6	80.0%	95.0%	95.0%	100.0%	80.0%	80.0%	80.0%	90.0%	90.0%	95.0%	80.0%	100.0%	88.5%
P#7	77.8%	66.7%	72.2%	72.2%	72.2%	77.8%	72.2%	72.2%	77.8%	38.9%	38.9%	77.8%	70.0%
Average											56%	93%	78%

Although the ART technique satisfied in total a coverage ratio better than the GA technique, the GA technique satisfied a coverage ratio close to the coverage ratio done by the ART technique for most subject programs except the fourth subject programs where the ART technique satisfied 92.0% and the GA technique satisfied 58%. Consequently, both the GA technique and the ART technique can create high quality test-suites which can satisfy high coverage ratio of all du-pairs of the subject programs.

VI. RESULTS SUMMARY

The results of the experiments showed that the GA technique defeated the ORT technique and the ART technique in reducing the size of the required test-suite to satisfy all du-pairs criterion. Where the GA technique creates in

total 31532 test-inputs to cover 74% of all du-pairs of the subject-programs with an average 4504.6 while the ART technique generates 61841 test-inputs to cover 78% of all du-pairs with an average 8834.4 and the ORT technique generates 32064 test-inputs to cover 73% of all du-pairs with an average 4580.6. Besides, the GA technique generates for each subject program a test-suite smaller than the test-suite generated by the ORT technique and the ART technique.

Further, the results showed that the GA technique defeated the ORT technique and the ART technique in reducing the number of generations required to cover all du-pairs criterion. Consequently, the GA technique converges faster than the ORT technique and the ART technique. Where the procedure of the GA technique was repeated in total 3153 times

to cover 74% of all du-pairs with an average 450.5 times while the procedure of the ART technique was iterated 6184 times to cover 78% of all du-pairs with an average 883.4 times and the procedure of the ORT technique was repeated 3206 times to cover 73% of all du-pairs with an average 458.1 times. Besides, the convergence rate of GA-based technique = 8.25 generations/second, the convergence rate of the ORT = 11.98 generations/second, and the convergence rate of the ART = 13.27 generations/second.

Moreover, the results showed that the GA technique is faster than the ART technique and slower than the ORT technique. Where the GA technique consumed in total 382 seconds to cover 74% of all du-pairs with an average 54.6 seconds while the ART technique consumed 465.9 seconds to cover 78% of all du-pairs with an average 66.6 seconds and the ORT technique consumed 267.6 seconds to cover 73% of all du-pairs with an average 38.2 seconds.

Additionally, the results showed that the coverage ratio of all du-pairs criterion done by the ART technique is higher than the coverage ratio of du-pairs done by the GA technique and the ORT technique. Where the GA technique satisfied coverage ratio of all du-pairs equals 74% while the ART technique satisfied coverage ratio of all du-pairs equals 78% and the ORT technique satisfied coverage ratio of all du-pairs equals 73%.

VII. CONCLUSION

This paper introduced an empirical comparison for genetic algorithms and adaptive random techniques according to four factors: reducing the size of the test-suite, convergence speed, elapsed time, and the effectiveness of maximizing the coverage ratio of all du-pairs criterion. A set of experiments was conducted to assess the two techniques according to the four factors. The results of the experiments showed that the GA technique defeated the ORT technique and the ART technique in reducing the size of the required test-suite to satisfy all du-pairs criterion. Further, the results showed that the GA technique defeated the ORT technique and the ART technique in reducing the number of generations required to cover all du-pairs criterion. Consequently, the GA technique converges faster than the ORT technique and the ART technique. Moreover, the results showed that the GA technique is faster than the ART technique and slower than the ORT technique. Additionally, the results showed that the coverage ratio of all du-pairs criterion done by the ART technique is higher than the coverage ratio of du-pairs done by the GA technique and the ORT technique. From these results, we concluded that GA algorithms are more effective than ORT and ART techniques in data-flow testing but ART satisfied the most coverage ratio. Therefore, we recommend hybridizing GA and ART and applying the hybrid technique in data-flow testing. Therefore, the future work will focus on hybridizing the genetic algorithms and the adaptive random techniques and applying the hybrid technique in the test-data generation process.

ACKNOWLEDGEMENT

The authors gratefully acknowledge Qassim University represented by the Deanship of Scientific Research on the material support for this research under the number (2018-1-14-S-3997) during the academic year 1440 AH/2018 AD.

REFERENCES

- [1] G. J. Myers, C. Sandler, and T. Badgett, *The Art of Software Testing*, 3rd ed. Hoboken, NJ, USA: Wiley, 2011.
- [2] I. Burnstein, *Practical Software Testing: A Process-Oriented Approach*, 1st ed. New York, NY, USA: Springer-Verlag, 2003.
- [3] S. Mahmood, "A systematic review of automated test data generation techniques," M.S. thesis, School Eng. Blekinge Inst. Technol., Karlskrona, Sweden, 2007.
- [4] J. Voas, L. Morell, and K. Miller, "Predicting where faults can hide from testing," *IEEE Softw.*, vol. 8, no. 2, pp. 41–48, Mar. 1991.
- [5] M. R. Girgis, "Using symbolic execution and data flow criteria to aid test data selection," *Softw. Test., Verification Rel.*, vol. 3, no. 2, pp. 101–112, Jun. 1993.
- [6] A. S. Ghiduk, "On symbolic execution software testing," *Int. J. Inform. Med. Data Process.*, vol. 1, no. 1, pp. 38–49, 2016.
- [7] R. Ferguson and B. Korel, "The chaining approach for software test data generation," *Trans. Softw. Eng. Methodol.*, vol. 5, no. 1, pp. 63–86, Jan. 1996.
- [8] L. Zhang, T. Xie, L. Zhang, N. Tillmann, J. De Halleux, and H. Mei, "Test generation via dynamic symbolic execution for mutation testing," in *Proc. IEEE Int. Conf. Softw. Maintenance*, Sep. 2010, pp. 1–10.
- [9] A. S. Ghiduk, M. J. Harrold, and M. R. Girgis, "Using genetic algorithms to aid test-data generation for data-flow coverage," in *Proc. 14th Asia-Pacific Softw. Eng. Conf.*, Dec. 2007, pp. 41–48.
- [10] A. M. Khamis, M. R. Girgis, and A. S. Ghiduk, "Automatic software test data generation for spanning sets coverage using genetic algorithms," *Comput. Inform.*, vol. 26, no. 4, pp. 383–401, 2007.
- [11] D. N. Thi, V. D. Hieu, and N. V. Ha, "A technique for generating test data using genetic algorithm," in *Proc. Int. Conf. Adv. Comput. Appl. (ACOMP)*, Nov. 2016, pp. 67–73.
- [12] A. S. Ghiduk, "A new software data-flow testing approach via ant colony algorithms," *Universal J. Comput. Sci. Eng. Technol.*, vol. 1, no. 1, pp. 64–72, 2010.
- [13] T. Y. Chen, H. Leung, and I. K. Mak, "Adaptive random testing," in *Advances in Computer Science—ASIAN. Higher-Level Decision Making* (Lecture Notes in Computer Science), vol. 3321, M. J. Maher, Ed. Berlin, Germany: Springer, 2004.
- [14] T. Y. Chen, "Adaptive random testing," in *Proc. 8th Int. Conf. Qual. Softw., (QSIC)*, 2008, p. 443.
- [15] T. Y. Chen, F.-C. Kuo, R. G. Merkel, and T. Tse, "Adaptive random testing: The ART of test case diversity," *J. Syst. Softw.*, vol. 83, no. 1, pp. 60–66, Jan. 2010.
- [16] E. Nikravan, F. Feysi, and S. Parsa, "Enhancing path-oriented test data generation using adaptive random testing techniques," in *Proc. 2nd Int. Conf. Knowl.-Based Eng. Innov. (KBEI)*, Nov. 2015, pp. 510–513.
- [17] R. Huang, J. Chen, and Y. Lu, "Adaptive random testing with combinatorial input domain," *Sci. World J.*, vol. 2014, Mar. 2014, Art. no. 843248.
- [18] J. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI, USA: Univ. Michigan Press, 1975.
- [19] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd ed. Berlin, Germany: Springer-Verlag, 1999.
- [20] L. Cockrell, "On adaptive random search techniques," in *Proc. 7th Symp. Adapt. Processes*, Dec. 1968, p. 64.
- [21] M. S. Hecht, *Flow Analysis of Computer Programs*. New York, NY, USA: Elsevier, 1977.
- [22] S. Rapps and E. J. Weyuker, "Data flow analysis techniques for test data selection," in *Proc. 6th Int. Conf. Softw. Eng.*, Los Alamitos, CA, USA, Sep. 1982, pp. 272–278.
- [23] P. M. Herman, "A data flow analysis approach to program testing," *Austral. Comput. J.*, vol. 8, no. 3, pp. 92–96, 1976.
- [24] S. Rapps and E. Weyuker, "Selecting software test data using data flow information," *IEEE Trans. Softw. Eng.*, vol. SE-11, no. 4, pp. 367–375, Apr. 1985.
- [25] F. E. Allen and J. Cocke, "A program data flow analysis procedure," *Commun. ACM*, vol. 19, no. 3, p. 137, Mar. 1976.

- [26] A. S. Ghiduk and M. R. Girgis, "Using genetic algorithms and dominance concepts for generating reduced test data," *Informatica*, vol. 34, no. 3, pp. 377–385, 2010.
- [27] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE Trans. Softw. Eng.*, vol. 37, no. 5, pp. 649–678, Sep. 2011.
- [28] W. B. Langdon, M. Harman, and Y. Jia, "Efficient multi-objective higher order mutation testing with genetic programming," *J. Syst. Softw.*, vol. 83, no. 12, pp. 2416–2430, Dec. 2010.
- [29] P. May, J. Timmis, and K. Mander, "Immune and evolutionary approaches to software mutation testing," in *Artificial Immune Systems (Lecture Notes in Computer Science)*, vol. 4628, L. N. de Castro, F. J. Von Zuben, and H. Knidel, Eds. Berlin, Germany: Springer, 2007.
- [30] M. Polo, M. Piattini, and I. García-Rodríguez, "Decreasing the cost of mutation testing with second-order mutants," *Softw. Test., Verification Rel.*, vol. 19, no. 2, pp. 111–131, Jun. 2009.
- [31] A. S. Ghiduk and M. Rokaya, "An empirical evaluation of the subtlety of the data-flow based higher-order mutants," *J. Theor. Appl. Inf. Technol.*, vol. 97, no. 15, pp. 4061–4074, 2019.
- [32] A. S. Ghiduk, M. R. Girgis, and M. H. Shehata, "Employing dynamic symbolic execution for equivalent mutant detection," *IEEE Access*, vol. 7, pp. 163767–163777, 2019.
- [33] C. Michael, G. Mcgraw, and M. Schatz, "Generating software test data by evolution," *IEEE Trans. Softw. Eng.*, vol. 27, no. 12, pp. 1085–1110, Dec. 2001.
- [34] R. P. Pargas, M. J. Harrold, and R. R. Peck, "Test-data generation using genetic algorithms," *Softw. Test., Verification Rel.*, vol. 9, pp. 263–282, 1999.
- [35] A. S. Ghiduk, "Reducing the number of higher-order mutants with the aid of data flow," *e-Inform. Softw. Eng. J.*, vol. 10, no. 1, pp. 31–49, 2016.
- [36] A. S. Ghiduk, "Using evolutionary algorithms for higher-order mutation testing," *Int. J. Comput. Sci.*, vol. 11, no. 2, pp. 93–104, Mar. 2014.
- [37] D. S. Rodrigues, M. E. Delamaro, C. G. Corrêa, and F. L. S. Nunes, "Using genetic algorithms in test data generation: A critical systematic mapping," *ACM Comput. Surv.*, vol. 51, no. 2, p. 41, 2018.
- [38] K. E. Serdyukov and T. V. Avdeenko, "Using genetic algorithm for generating optimal data sets to automatic testing the program code," in *Proc. Int. Conf. Inf. Technol. Nanotechnol. (ITNT)*, 2019, pp. 173–182.
- [39] S. Rani, B. Suri, and R. Goyal, "On the effectiveness of using elitist genetic algorithm in mutation testing," *Symmetry*, vol. 11, no. 9, 1145, pp. 1–26, 2019.
- [40] B. Jiang, Z. Zhang, W. K. Chan, and T. H. Tse, "Adaptive random test case prioritization," in *Proc. IEEE/ACM Int. Conf. Automated Softw. Eng.*, Nov. 2009, pp. 233–244.
- [41] B. Jiang and W. Chan, "Input-based adaptive randomized test case prioritization: A local beam search approach," *J. Syst. Softw.*, vol. 105, pp. 91–106, Jul. 2015.
- [42] R. Huang, W. Sun, Y. Xu, H. Chen, D. Towe, and X. Xia, "A survey on adaptive random testing," *IEEE Trans. Softw. Eng.*, to be published.



FAHAD M. ALMANSOUR received the bachelor's degree in primary education from Qassim University, Saudi Arabia, in 2006, the M.S. degree in IT from De Montfort University, U.K., in 2010, and the Ph.D. degree in computing in programming development from Plymouth University, U.K., in 2017. He is currently an Assistant Professor with the Computer Science Department, Qassim University, Saudi Arabia. He is currently the Dean of Information Technology Deanship of Qassim University. His research interests include software testing, software usability, and software development.



ROOBAEA ALROOBAEA received bachelor's degree (Hons.) in computer science from King Abdulaziz University (KAU), Saudi Arabia, in 2008, and the master's degree in information system and the Ph.D. degree in computer science from the University of East Anglia, U.K., in 2012 and 2016, respectively. He is currently an Assistant Professor with the College of Computers and Information Technology, Taif University, Saudi Arabia. His research interests include human-computer interaction, software engendering, cloud computing, the Internet of Thing, artificial intelligent, and machine learning.



AHMED S. GHIDUK received the B.Sc. degree from Cairo University (Beni-Suef Branch), Egypt, in 1994, the M.Sc. degree from Minia University, Egypt, in 2001, and the joint Ph.D. degree from Beni-Suef University and the College of Computing, Georgia Institute of Technology, Atlanta, GA, USA, in 2007. He is currently an Associate Professor with the Department of Mathematics and Computer Science, Faculty of Science, Beni-Suef University, Egypt. He is also an Associate Professor with the College of Computers and Information Technology, Taif University, Saudi Arabia. His research interests include software engineering, search-based software engineering, software testing, mutation testing, higher-order mutation testing, weak mutation testing, test data generation, requirements engineering, and genetic algorithms.

• • •