On FSM-Based Testing

An Empirical Study: Complete Round-Trip Versus Transition Trees

Hoda Khalil
Department of Systems and Computer Engineering
Carleton University
Ottawa, Canada
hodakhalil@cmail.carleton.ca

Yvan Labiche
Department of Systems and Computer Engineering
Carleton University
Ottawa, Canada
labiche@sce.carleton.ca

***Abstract*—Finite state machines being intuitively understandable and suitable for modeling in many domains, they are adopted by many software designers. Therefore, testing systems that are modeled with state machines has received genuine attention. Among the studied testing strategies are complete round-trip paths and transition trees that cover round-trip paths in a piece wise manner. We present an empirical study that aims at comparing the effectiveness of the complete round-trip paths test suites to the transition trees test suites in one hand, and comparing the effectiveness of the different techniques used to generate transition trees (breadth first traversal, depth first traversal, and random traversal) on the other hand. We also compare the effectiveness of all the testing trees generated using each single traversal criterion. This is done through conducting an empirical evaluation using four case studies from different domains. Effectiveness is evaluated with mutants. Experimental results are presented and analyzed.**

Keywords— finite state machines; round-trip paths; transition trees; transition paths coverage.

I. INTRODUCTION

Finite state machines (FSMs) are particularly useful for modeling and testing various pieces of software [4, 5, 15, 30, 31]. Essential background information about FSMs and their testing techniques is explained in section II.

Chow [10] and Binder [8] are two main contributors in state based testing. Chow introduced the W-method that generates test suites using a transition tree traversal of the FSM representing the system under test (SUT). The collection of paths composing the tree is then used as a collection of test cases. Binder later introduced the concept of *round-trip* paths [8], paths that are prime paths of non-zero length that start and end at the same node [1]. A path from node n_i to node n_j is a *prime path* if it is a simple path and it does not appear as a proper sub path of any other simple path [1]. A *simple path* does not contain any repeating node, except possibly the first and last nodes of the path [8]. However, Binder covers the round-trip paths in pieces by adapting the transition tree solution of Chow [10]. The work of Chow and Binder among other research literature is summarized in section III. Binder's way of covering round-trip paths in his N^+ method raises the first of the four research questions (section IV): is covering complete round-trip paths equivalent to covering them in pieces using transition trees? Binder argues that either a breadth first search (BFS) or depth first search (DFS) can be used to create a tree and generate round-trip paths. This triggers our second research question: Are BFS and DFS trees different from a testing point of view. Last, since there are typically several BFS/DFS trees for an FSM, we ask: Are all BFS (resp. DFS) trees equivalent from a testing point of view?

Our interest in these techniques stems from the fact that the transition tree criterion was shown to be a good compromise among other FSM testing criteria, and trees have been claimed to be equivalent to complete round-trip paths testing. However, there is anecdotal evidence trees are not equally effective at finding faults. In this paper, we provide a more systematic study to compare the transition tree criterion and the round-trip path on one hand, and compare the different transition trees generated on the other hand.

To conduct the experimental comparisons, we use four case studies (section V), generate all possible BFS and DFS trees following Binder/Chow's algorithms, as well as random test suites and test suites that exercise round-trip paths in their entirety. To facilitate experiments, we deployed an automation tool chain [22], which we briefly describe together with the general experimentation setup (section VI). We compare test suites in terms of cost and fault (mutant) detection. Results in section VII lead us to develop some hypotheses, which we validate in section VIII. We discuss threats to validity in section IX, and conclude in section X.

II. BACKGROUND

In this section, we first discuss the notion of an FSM and then briefly discuss the notion of FSM-based testing;

A. FSM

The term finite automata was first introduced as a mathematical concept to model hardware systems such as sequential circuits [29]. FSMs are the engineering application of finite automata [7]. An FSM is one of the most powerful ways to represent a control system where a number of stimuli (inputs) is received from the application and actions (outputs) are produced to affect the application [34]. State machines are used to describe behaviors of sequential systems where outputs depend on inputs and the current state. This is to be opposed to combinational sequences where the output is only dependent on the set of inputs.

States, transitions, events, and actions are the building blocks of an FSM [7]. The collection of states represents all the possible situations in which the FSM maybe. The system goes through a sequence of events to reach a certain state. A state is some kind of a memory that represents the history of the model [34]. From a software point of view, a state is a set of specific values for a collection of variables [1]. A transition is an allowable two-state sequence that may result in an output action and must specify an accepting state and a resultant state [7]. A transition occurs in zero time, and usually means a change in the value for one or more variables [1]. An event can be an input or an interval of time that triggers a transition [7]. Actions associated with transitions are the result or the output produced when an event triggers the transition [7].

We use transition diagrams, among other possibilities, to represent FSMs, since they are usually the most used state machine representation [34]. In a transition diagram, states are represented by nodes or simply circles, while transitions are mapped into arcs. The tail of the arc is at the source state, while the head is at the destinations state.

B. *FSM-based Testing*

FSM-based testing, or simply FSM testing, is a kind of model-based testing [33]. Model-based testing is a black box testing technique based on a formal model that is usually built to facilitate automation [28, 30]. FSM testing has been widely covered in the testing literature [1]. During FSM testing, one creates test paths, which are sequences of states and transitions, to satisfy one (or more) selection criterion. A large number of criteria exist for FSM testing [1, 6, 8, 22, 24]. Since the focus of this paper is on the round-trip path criterion, we limit our discussion of related work on criteria that relate to round-trip paths. We use Binder classification as a primary source for the discussion [8]. The all round-trip paths criterion is satisfied when all paths that represent loops are exercised. The M-length signature strategy assumes that the SUT is opaque, i.e. that there is no means to determine the current state. So, to conclude that the SUT is in a certain state, a signature or a distinguishing sequence of inputs that produces different outputs for each state is applied. The most classic method based on this criterion is Chow's W-method [10]. The n-switch criterion is satisfied when all sequences of n consecutive transitions are exercised. A 0-switch refers to the all-transitions criterion, and 1-switch refers to exercising all sequences of two transitions, which is often called transition-pairs.

III. Related Work

Chow's method [10] is the foundation of many other state based testing strategies. The first step of the method generates a transition tree from the graph representing the state diagram. The procedure for generating the tree mainly states that each node is branched only the first time the traversal of the FSM hits that node. There could be more than one transition tree for the same FSM. Each path in the transition tree, from the root to a leaf node, is considered a test case. Given the nature of those test paths, Binder refers to the technique as the round-trip path technique (RTP) arguing that the tree covers round trip paths (recall section II.B) although not all RTPs are covered in their entirety by the tree and some are covered in pieces [21, 25]. However, state-based testing cannot be effective without the ability of determining the resultant state after applying each test sequence (oracle). In the second step, a characterization sequence is appended to each test case thereby assuming the system is opaque. When the software is not opaque, Binder suggests to have elements in its API to get and set information about the state of the implementation [8]. The tree generation procedure is essentially a BFS traversal of the FSM graph and Binder mentions that a DFS traversal is also possible. Binder claims that the structure of the tree, either BFS or DFS, should not make a difference in producing the round-trip paths, since in all cases, all round-trip paths will be included in the resulting tree (though possibly in pieces). He however mentions that different trees may have different effectiveness at finding faults. This is essentially the assertion we want to validate.

Several studies empirically compare different state-based testing criteria similarly to what we propose. We discuss the most relevant ones, and that are specifically concerned with the RTP criterion.

Holt and colleagues [17], study the cost-effectiveness impact of some selection criteria, test model and test oracle. The evaluation is done using 26 real faults applied manually to a real case study representing a safety monitoring component in a control system. The criteria evaluated are all transitions, all round-trip paths, all transition pairs, paths of length two, paths of length three, and paths of length four. Results show that all-transition pairs is the most expensive criterion while all-RTPs provides the best tradeoff between cost and fault detection followed by all paths of length three.

Khalil and Labiche [24] study, with two case studies, whether the traversal trees generated using the W-method are equivalent in terms of cost and effectiveness, though not in a systematic way as we report next. The authors study BFS traversal, DFS traversal, as well as a novel graph traversal algorithm. The research concludes that the effectiveness of transition trees varies, but with low statistical significance. However, further experimentation is required to generalize the findings and compare tree generation techniques.

As clarified by the above mentioned related work, Chow's method (transition trees), adapted by Binder has been shown to be a cost-effective alternative to cheap but ineffective all-transitions and effective but expensive all-transition-pairs [7, 15, 21]. This is a strong motivation for conducting our empirical study. We extend previous work [24] with two new case study systems, but more importantly by systematically generating all possible trees; we analyze a total of 102,113 test suites as opposed to only 12 in the previous study. We aim to evaluate the different approaches for developing adequate test suites for the transition trees/RTP criterion for the purpose of maximizing the effectiveness of the studied criterion.

IV. Research Questions

In light of the previous discussions, our study aims at answering four research questions.

RQ1: Are all the mutants revealed by a round-trip paths test suite that exercise those paths entirely also revealed by test suites derived from transition trees of the same FSM diagram?

RQ2: Does the algorithm used (BFS, DFS, or Random) to generate the transition tree affect mutation score?

RQ3: Do distinct trees generated using one algorithm differ in their effectiveness at finding faults?

RQ4: Is it possible to derive a common trend in the trees generated that helps increase the mutation score?

V. Case Studies

Determining the effectiveness of the test criteria mentioned earlier cannot be performed by analytical means only. Like other types of criteria (e.g., data flow criteria [13]), experimental evaluation is required. Case study research is an exploratory method well suited for many kinds of software engineering research [30]. The five steps for conducting a case study are: case study objective definition (section IV) and design (section VI), preparation for data collection, collecting evidence, analysis of collected data, and reporting (section VII).

The four case studies used in our experiments are a data structure representing an ordered set, an embedded controller represented by a cruise control simulator, an automatic telling machine (ATM), and an electromechanical device represented by videocassette recorder (VCR). The four case studies are implemented in Java. The cruise control case study is originally described by Gomaa [15]. The ATM code is first generated using the State Machine Compiler [32]. The VCR case study is based on work done by Q. Lin [26], while the ordered set is used by others [21, 25] and is available from the Software Artifact Infrastructure Repository [11]. Table 1 summarizes structural characteristics of each case study. The FSMs can be found elsewhere [23]. The cruise control is the smallest case study, while the ATM and the ordered set have a medium size. The VCR case study is the largest.

Table 1 Case Studies Characteristics

Case Study Name	Cruise Control	ATM	Ordered Set	VCR
No. of States	5	10	9	17
No. of Transitions	29	22	35	64
Lines of Code	211	473	273	1493

VI. Experimental Setup

The experimentation process is composed of four main steps. The first step is to generate the test suites based on the four algorithms previously mentioned (complete round trip, BFS, DFS, and random). The second step is to produce JUnit files that can run the test suites. The JUnit files also use test oracles to verify that the current state of the SUT is the expected one. The third step of the experiment is seeding the code with mutants. The fourth and final step is running the test suites against mutants.

To conduct our study, automation is paramount. Alternatively, one can produce them manually though this is very time consuming, likely error-prone, and sometimes nearly impossible: e.g., the VCR FSM leads to 101,947 unique adequate test suites for the DFS technique. Similarly, executions and mutant generation must be automated. We therefore developed the State-based Test Suites Generation and Execution (STAGE) framework tool chain [22].

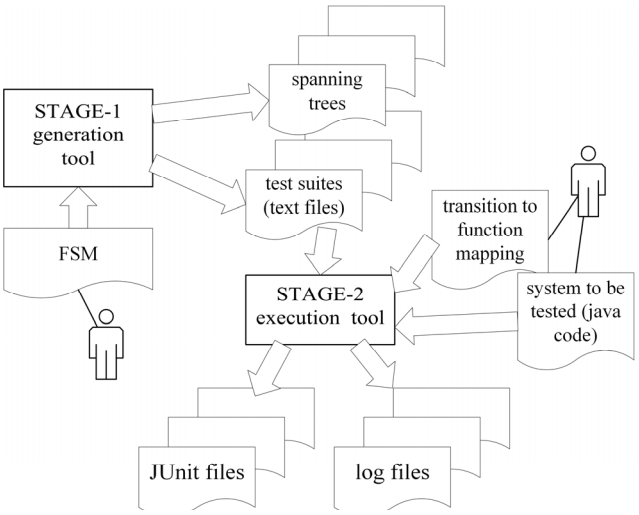


Fig. 1. STAGE overview

The tool chain consists of two tools STAGE-1 and STAGE-2: Fig. 1. STAGE-1 gives the user the option to produce test suites according to different state-based adequacy criteria; currently supporting random, round-trip, BFS, or DFS graph traversal algorithms. Two remarks about the algorithms are worth mentioning. First, the random test suites generation algorithm is an unbiased algorithm that can generate trees with paths as lengthy as those of the DFS trees and as short as those of the BFS trees [23]. Second, the round-trip paths algorithm chooses, for each RTP, the shortest prefix from the initial state of the FSM to reach the start state of the RTP in order to minimize the prefix contribution to the mutation score. Parameters (for triggering events) are then added where needed by the test engineer to the test suites produced by STAGE-1. For instance, in the case of the ordered set case study, which requires that events have integer parameters, we try to randomly select integer values while making sure insertions in the (ordered) data structure take place at the beginning, end and middle of the data structure in each test suite. The test engineer also provides to STAGE-2 the mapping between the transitions in the FSM and the corresponding functions to call in the SUT. STAGE-2 automatically executes test suites using this mapping. It also encloses a test oracle that is used after executing each function/transition. STAGE-2 can also generate JUnit tests instead of directly executing the test suites. This is useful, as the JUnit tests may be used by other tools to run the test suites, which is the case with the Major mutation tool used in our experiments [21].

To compare the different testing strategies, we use fault detection effectiveness, specifically mutation analysis. Mutation analysis can be used to evaluate the quality of test cases [1]. Although a reliable method [2, 3, 17], mutation analysis is time consuming [21]. We selected Major [21] to automatically seed mutants since it uses fewer mutation operators than other mutation tools to avoid the creation of many easy to find mutants. Having many easy to find mutants

complicates the task of comparing the different test suites since it is likely that most test suites will detect the easy to find mutants and therefore, the difference between the mutation scores of the different test suites will be too small to be studied with precision. Also, Major integrates well with JUnit. Major has other advantages not described here. Last, Major determines mutation coverage, that is the number of mutants which are hit (i.e., the line of code where they are seeded is executed) by test suites. Major also calculates the mutation score which is the ratio of killed mutants over the total number of unique mutants [1]. This ratio gives an evaluation of the fault revealing power of a test suite [33].

By measuring both mutation coverage and mutation score, Major is identifying equivalent mutants; those are mutants that are syntactically different from the original program, but not semantically different. Therefore, they cannot be detected or killed by the test suites although they are covered by them [19]. Although some covered but alive mutants may not be equivalent, as this depends on the tests we use, we consider a mutant that is covered and alive with all our test suites as equivalent. Given the large, to very large number of test suites we experiment with, we believe this is a reasonable assumption. Plus, since we compare test suites between one another, this assumption should not introduce a significant threat to validity. In our experiments, we use all the mutation operators available with Major.

When measuring mutation score for an RTP test suite, i.e., a test suite whose test cases are made of a prefix and a complete RTP, we measure the cumulative mutation score of the prefix and the RTP parts of tests. Our prefixes being small in length, in fact smaller than RTPs, mutation score of those tests is primarily the RTPs' contribution. Future work will look into measuring RTPs' contribution separately.

VII. Results

The numbers of unique test suites generated by STAGE-1 and the average number of transitions executed per test suite for each traversal algorithm are shown in Table 2. We discuss next each case study separately. For each case study, we produce a graph that reports for each of the four main test suite generation strategies (BFS, DFS, random, and RTP) the mutation score, the ratio of mutation score over mutation coverage, mutation coverage as well as average execution time.

Table 2 STAGE-1 Output Matrix

Case Study Name		Cruise Control	ATM	Ordered Set	VCR
BFS Trees	*No. of Trees*	3	9	4	24
	Avg. No. of Transitions/ Test Suite	89	53	61	174
DFS Trees	*No. of Trees*	3	27	59	101,947
	Avg. No. of Transitions/ Test Suite	89	61	81.9	293.42
RTP	*No. of Transitions*	225	84	73	9768
Random Trees	*No. of Trees*	10	10	10	10
	Avg. No. of Transitions/ Test Suite	89	58.6	67	201.6
Generated Mutants		336	323	727	1649

A. Cruise Control

As shown in Table 2, STAGE-1 produces three BFS test suites, three DFS test suites, ten random test suites, and one RTP test suite. Due to the structure of the FSM, the BFS and DFS test suites are identical. 149 (44.35%) mutants are covered by each BFS/DFS test suite. Two of these test suites kill 58 mutants, while one kills 59 mutants (17.3%): Fig. 2 (a). The different BFS/DFS traversals do not affect mutation score, neither do they affect mutation coverage. Nothing in the structure of the trees can explain the different execution times; we conjecture this is due to caching. The RTP test suite kills 59 mutants out of 149 covered mutants. This gives mutation score of 17.5% and coverage ratio of 44.3%. Only one of these 59 mutants is not killed by any other test suite. There is no clear difference between BFS, DFS and RTP, though RTP is roughly twice as expensive as BFS and DFS

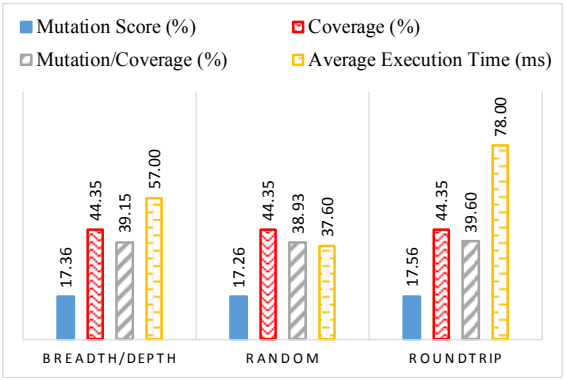
One general observation we can make is the low mutation score of all test suites: 17%. This can be explained by the fact that many of the seeded faults are related to the timing aspect of the code (e.g., speed increases when the driver pushes on the gas pedal), which is not handled in the FSM (the FSM is an abstraction).

B. ATM

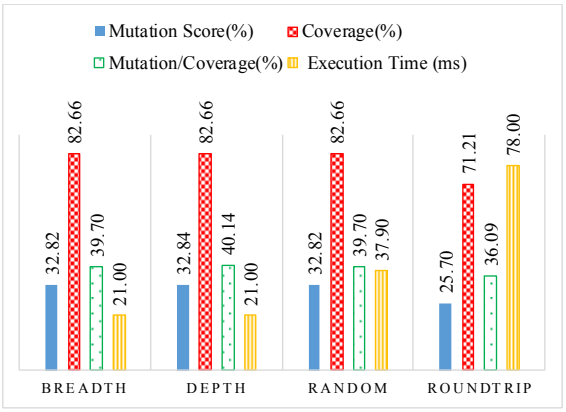
STAGE-1 generates nine BFS test suites, 27 DFS test suites, ten random test suites, and one RTP test suite (Table 2). From Fig. 2 (b) we observe that 106 (32.8%) of the seeded faults are detected by all BFS, DFS, and random test suites. The RTP test suite detects only 83 mutants (25.7%). For coverage, all test suites except for RTP cover 267 mutants (82.6%), while the RTP test suite covers 230 of the seeded mutants (71.2%).

The reason for the lower mutation score of the RTP is that the round-trip paths do not exercise all the transitions. Binder's solution to this issue is to recommend the execution of simple paths [8]. Since we are primarily interested in RTPs, we did not include simple paths in our test suites. However, the ratio mutation score to coverage still shows lower measure for the RTP test suite than other test suites: some mutants are not killed even if they are covered; specific paths in the FSM need to be triggered in order to kill the mutants and those paths are not exercised by the RTP (even with prefixes). We also note that all the mutants killed by the RTP test suite are killed by the other test suites.

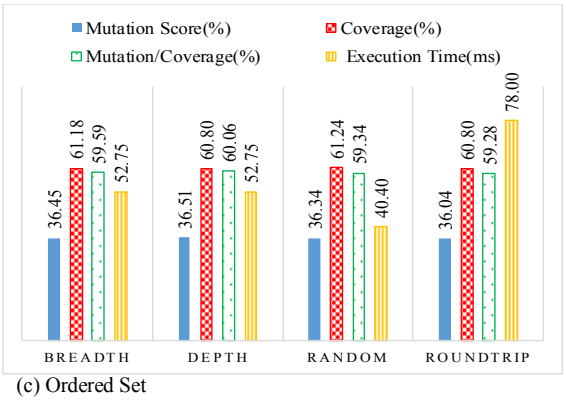
This is a different observation than the case in the cruise control case study, which can be explained by looking at the FSMs. There is only one possible shortest prefix to each round-trip path in the cruise control FSM. This means that the RTP test suite in the cruise control is not missing any transition by choosing the shortest prefix since there is only one prefix unlike the case with the ATM FSM. In other words, all states except for the initial state are members of at least one round-trip path. This leads to expecting higher mutation score from the RTP test suite of the cruise control case study unlike the case with the ATM. The RTP test suite takes the longest to execute: Fig. 2 (b).



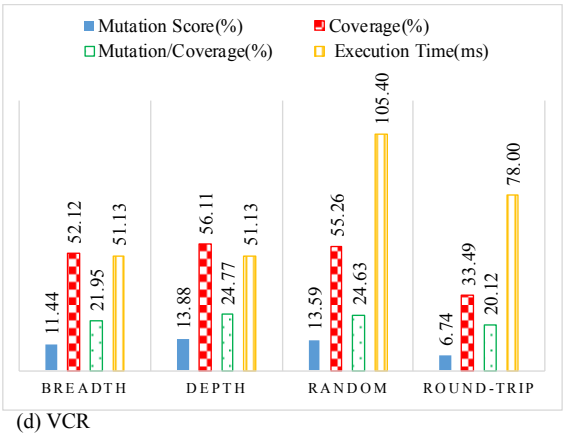
(a) Cruise Control



(b) ATM



(c) Ordered Set



(d) VCR

Fig. 2. Test Suites Results (Mean Values) for All Four Case Studies

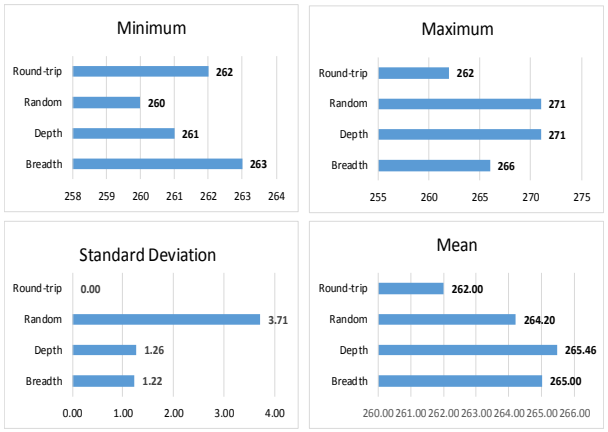


Fig. 3. Ordered Set Killed Mutants Results.

C. *Ordered Set*

STAGE-1 generates four BFS test suites, 59 DFS test suites, ten random test suites, and one RTP test suite (Table 2) for the ordered set case study. From Fig. 2 (c), the number of mutants killed for the BFS test suites ranges from 263 to 266 mutants out of 727, while coverage ranges from 443 to 450 mutants. For the DFS test suites, the number of killed mutants ranges from 260 to 263 mutants, while coverage is 442 mutants for all test suites. For the random test suites, the number of killed mutants is between 260 to 271 mutants, while covered mutants are at a minimum of 442 and a maximum of 450 mutants. The number of killed mutants for the RTP test suite is 262 mutants, while the coverage is 442 mutants. The mean values of the number of killed mutants, mutation coverage, ratio between killed mutants and mutation coverage, and the execution time are shown in Fig. 2 (c).

The highest mean mutation score is achieved by the DFS test suites, while the highest coverage is achieved by a random as well as one of the BFS test suites. We note that the test suite that achieves the highest mutation score is not the one that achieves highest coverage.

Fig. 3 lists the mean, maximum, minimum, and standard deviation values for the number of killed mutants. Since the ordered-set FSM results are more varied than the two previous case study systems, more analysis of the results is needed. The reason for listing/plotting the maximum and minimum values is that spikes in mutation score might bias the average towards certain values. Therefore, ignoring the minimum, maximum and variation among the values while only looking at the mean might be misleading.

Although the BFS test suites have higher mutation coverage on average than the DFS test suites, Fig. 2 (c), the DFS test suites have better mutation score than the BFS test suites. The round-trip algorithm has the highest average execution time. The higher execution time of the round-trip is due to more and longer test cases in many cases. The random test suites also have the highest standard deviation in mutation score which is expected as the random nature of the algorithm results in trees that vary in structure. The mutation score and standard deviation of DFS test suites are slightly higher than that of BFS test suites. Fig. 4 shows how the effectiveness of

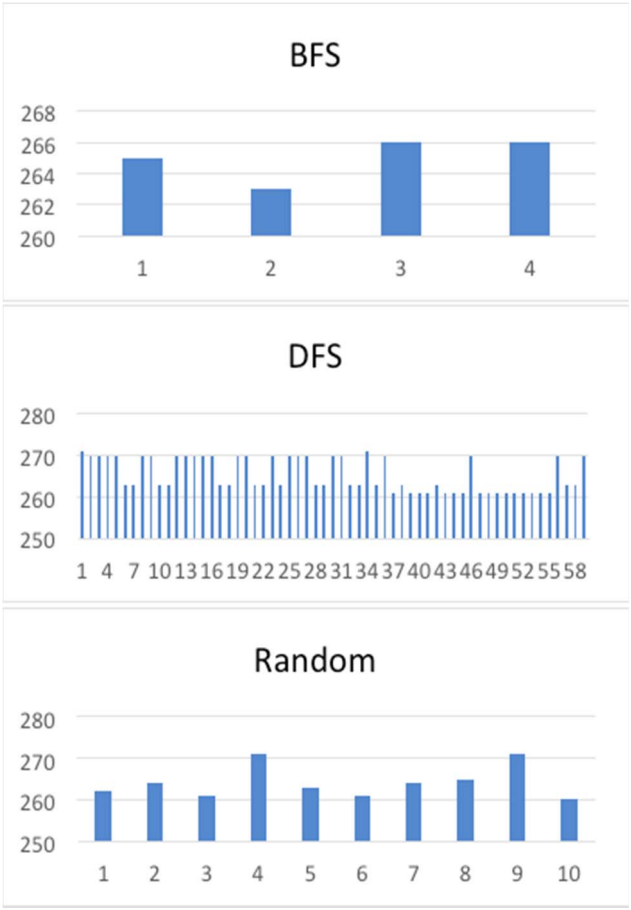


Fig. 4. Ordered Set Killed Mutants Comparison within Each Traversal Method (X-Axis: Test Suite No., Y-Axis: Killed Mutants.

finding mutants varies among different test suites belonging to the same traversal strategy of the ordered set FSM. For example, only two out of the four BFS test suites kill the same number of mutants.

D. VCR

Table 2 shows that STAGE-1 generates 24 BFS test suites, 101,947 DFS test suites, 10 random test suites, and one round-trip paths test suite for the VCR case study. Given the vast number of DFS test suites produced, we focus our analysis on 1,887 DFS test suites only. The 1,887 test suites span the complete set of DFS test suites; they are not clustered on the first few thousand for instance, and are rather sampled from the whole set of test suites. We tried to maintain a large difference between the structure of the chosen test suites. This guarantees that this sample is representative since the way the DFS algorithm is designed results in the fact that any two highly similar trees differ by only one transition.

Out of the 1469 seeded mutants, the BFS test suites catches between 150 and 206 mutants with an average of 168 mutants. The number of mutants revealed by the DFS test suites are between 192 and 215 mutants, while the random test suites are at a minimum of 192 and a maximum of 206 mutants killed. Finally, the RTP test suite kills 99 mutants.

Fig. 2 (d) compares the mean values of the killed mutants, coverage, mutation to coverage ratio, and execution time for all the algorithms. Test suites generated using the DFS algorithm have the highest mean mutation score (13.8%), directly followed by the random algorithm (13.5%), then the BFS (11.4%) and finally the lowest mutation score is for the RTP test suite (6.7%). The coverage follows the same order as the mutation score as shown in Fig. 2 (d). The random test suites are on average the slowest followed by the RTP, and then the BFS and DFS test suites are the fastest: Fig. 2 (d).

Fig. 5 visualizes the comparison between the different test suites in terms of killed mutants. It illustrates how the number of killed mutants varies between the different test suites, even if those test suites are generated using the same traversal algorithm (i.e. BFS, DFS, or Random). Although the DFS chart of Fig. 5 is crowded due to the number of exercised test suites (1887), it is clear from the height of the bar that the number of killed mutants varies between 190 and 215 killed mutants (see also Fig. 6).

We observe from the experiment that 27 mutants are killed only by the RTP test suite and the DFS test suites, but not killed by any other test suite. In addition, 18 mutants are killed only by the RTP test suite.

E. Results Analysis

In this section, we summarize experimental results for all the case study systems to answer the research questions proposed in section IV. Table 3 collects all the killed mutants statistical data of the four case studies. Maximum values among all algorithms for the number of killed mutants, mean, minimum, and maximum are underlined for each case study.

1) RQ1: Are all the mutants revealed by a complete round-trip paths test suite also revealed by test suites derived from transition trees of the same FSM diagram?

The answer to this question is simply no. In two of our case studies, the cruise control and the VCR, the complete round-trip paths test suite reveals faults that the other test suites covering round-trip paths in pieces do not reveal. In the case of cruise control, the complete RTP test suite reveals one fault that is not revealed by any other test suite. For the VCR, the results are more dramatic: The RTP test suite reveals 46 faults that neither the BFS test suites nor the random test suites manage to reveal. This is equal to approximately 21% of the highest mutation score achieved by all algorithms. However, only 27 of those 45 mutants are revealed by the DFS test suites.

2) RQ2: Does the algorithm used (BFS, DFS, or Random) to generate the transition tree from which the test suite is derived affect the mutation score?

From the data presented before and summarized in Table 3, the DFS test suites achieve the highest mutation score mean for the ordered set and the VCR case studies, while they share the highest score with the BFS traversal trees for the cruise control and the ATM case studies.

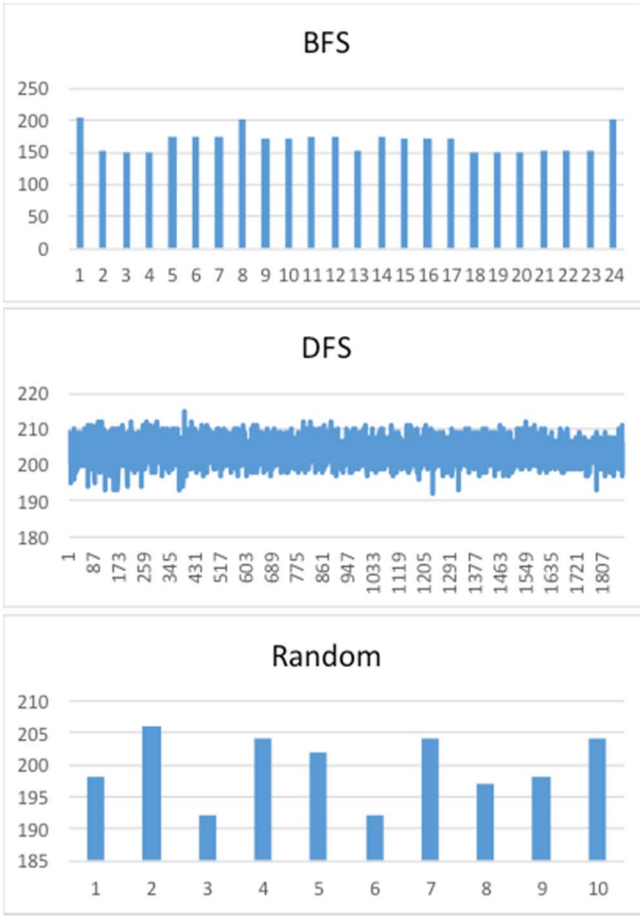


Fig. 5. VCR Killed Mutants Comparison Within Each Traversal Method (X-Axis: Test Suite No., Y-Axis: Score

The cruise control has identical DFS and BFS test suites as explained in section V. This neither refutes the argument in favor of using DFS test suites, neither supports it statistically.

For the ATM case study, when studying the different trees produced from STAGE-1, we observe that few partial paths (parts of longer paths) are common to all trees. The length of those paths contributes to almost half of the total paths length.

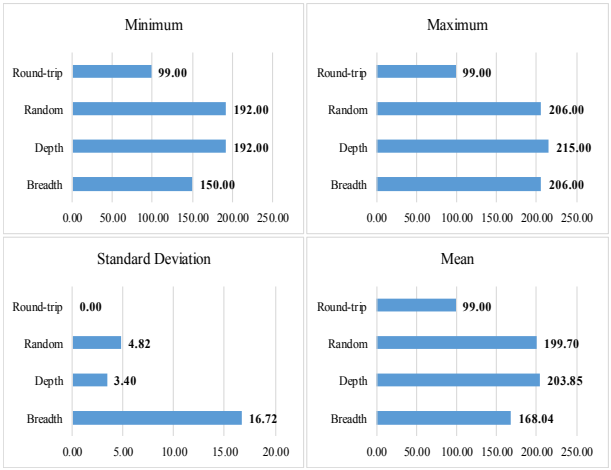


Fig. 6. VCR Killed Mutants Statistics

Thus, the resulting trees spread horizontally while having relatively small depth. The consequence of this is that when using either the BFS or the DFS algorithm, the differences between the trees are minimal.

More precisely, in all cases the difference between one tree and another is a replacement of one edge and its destination node. It may be concluded, on the one hand, that this is the reason why we observe the same mutation score achieved by BFS, DFS, and random algorithms as shown in Table 3. On the other hand, this same observation is the reason the RTP test suite achieves quite low mutation score. Recall each test in a RTP test suite is made of a shortest path prefix to a complete round-trip path. Hence such test suite misses all transitions that are neither part of the round-trip paths nor the prefixes added to these paths. In addition, the length of a round-trip path is quite short for ATM, since all the state composing round-trips form a strongly connected cluster, which is not the case with the ordered set, nor the VCR.

Although DFS test suites produce lower mutation score in the case of the ordered set case study, there is more than one reason to consider this piece of data an outlier that does not contradict the previous, general observation we made. First, the difference is only two mutants out of 263 and it is in only one DFS tree. Second, the standard deviation of the killed mutants of the DFS test suites is in most cases lower than the standard deviation of other test suites. In other words, the mutation score does not vary considerably for DFS test suites. This means that in most cases the average mutation score is a valid measure for the mutation score of the DFS test suites. This is especially evident in the two most significant case study systems: the ordered set and the VCR.

In brief, the experimental results suggest that the answer to RQ2 is yes: the algorithm used for tree traversal does affect mutation score and DFS seems to be the best alternative. Fig. 7. shows how the DFS trees mutation score exceed on average the rest of the test suites.

Table 3 Killed Mutants Results Summary for All Case Studies

No. Of Killed Mutants\Algorithm		BFS	DFS	Random	Round -trip
Mean	Cruise Control	58.3	58.3	34.7	59.0
	ATM	106.0	106.0	106.0	83.0
	Ordered Set	265.0	268.7	264.2	262.0
	VCR	168.0	203.9	199.7	99.0
Min.	Cruise Control	58	58	59	59
	ATM	106	106	106	83
	Ordered Set	263	261	260	262
	VCR	150	192	192	99
Max.	Cruise Control	59	58	58	59
	ATM	106	106	106	83
	Ordered Set	266	269	271	262
	VCR	206	215	206	99
Standard Deviation	Cruise Control	0.1	0.1	0.0	0.0
	ATM	0.0	0.0	0.0	0.0
	Ordered Set	1.2	1.3	3.7	0.0
	VCR	16.7	3.4	4.8	0.0

3) RQ3: Do the distinct trees generated using one algorithm differ in their effectiveness at finding faults (a.k.a. their mutation score)?

The most evident example is the BFS test suites of the VCR case study. While the minimum number of killed mutants achieved by BFS test suites is 150 mutants, the maximum is 206 mutants. This is also clear by the high standard deviation of the BFS test suites of this case study as indicated in the last row of Table 3. However, the overall trend is low standard deviation for test suites falling in the same group especially for the DFS test suites.

The test suites sometime vary significantly in terms of mutation score, even within one group of test suites, i.e., using the same traversal algorithm. Fig. 4 and Fig. 5 represent the variation in mutation score of the VCR test suites and the ordered set test suites, respectively. The charts illustrate how the effectiveness of test suites generated using the same strategy (BFS, DFS, or Random) can vary in terms of killing mutants.

4) RQ4: Is it possible to derive a common trend in the trees generated that helps increasing the mutation score?

Observing that the test suites generated using the DFS traversal algorithm perform better on average, we tend to believe that there is a positive correlation between the length of test cases (a DFS tree has longer test cases than a BFS tree) and the mutation score. We also compared manually trees with high mutation score with lower mutation score trees across the spectrum of the same traversal algorithm. It is observed that in many cases the tree (test suite) with better mutation score has lengthier paths (test cases). However, no statistical evidence can be deduced from this observation. We attempt to answer this question empirically in the following section, through hypotheses we attempt to validate with our data.

VIII. HYPOTHESES

In this section, in light of the results presented earlier, we put forward some hypotheses, which we attempt to validate with the data we collected.

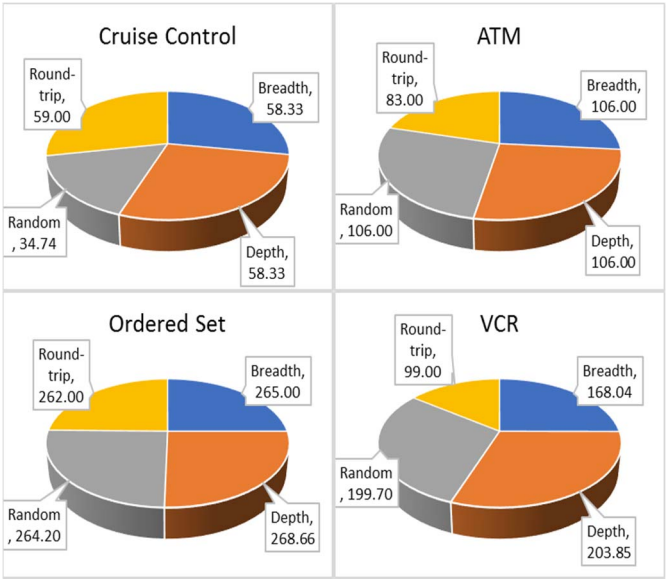


Fig. 7. Mean Killed Mutants Values

A. Hypothesis #1: Test Paths Length Affects Mutation Score

It is worth noting that several studies support the hypothesis that longer test cases are more effective at finding faults [4, 5], 14]. However, in the case of transition trees, increasing the length of a test case affects the structure of the rest of the tree and consequently the length of other test cases in the same test suite. Therefore, the mentioned hypothesis might not hold for the case of transition trees.

To prove the validity of this hypothesis, the mean paths length (MPL) has to be calculated for each test suite.

$$\text{MPL}=\left(\frac{sum\ of\ all\ paths\ lengths\ for\ all\ test\ cases\ in\ the\ test\ sutie}{number\ of\ test\ cases\ in\ the\ test\ suite}\right)$$

Given the considerable number of trees produced in the experiments, we have developed an automation tool to perform the calculations. For each case study, the average paths lengths are compared across all traversal algorithms. The mutation scores are plotted against the average lengths for each group of test suites derived using the same traversal algorithm. Also, one chart visualizing the relationship between the average paths lengths and the mutation score for all generated test suites regardless of the traversal algorithm is plotted.

1) Cruise Control

On the one hand, all test suites derived for the cruise control case study achieve the same mutation score, except for one test suite that misses only one of the faults revealed by the other test suites. On the other hand, all the test suites have the same mean paths length. The results of the calculations for the cruise control case study are in support of the suggested hypothesis if the minor difference in mutation score is neglected. However, one cannot use this result to generalize the hypothesis to other SUT without further experimentation and results analysis.

2) ATM

The ATM case study test suites achieve the same mutation score across all groups belonging to different traversal algorithms. Contrary to what is hypothesized, the average length of the test suites paths varies. The average paths length ranges from 8.57 to 10.86 for the test suites of the ATM case study (i.e., around two extra states per paths difference between the test suite with shorter average paths length and the one with longer average paths length).

3) Ordered Set

For the ordered set case study, there are some interesting observations. The average length is the same for all BFS test suites, while the number of killed mutants vary slightly between 263 to 266 mutants with these test suites. For the randomly generated test suites, the general trend is that the mutation score increases in proportion to MPL. For the DFS test suites, the number of killed mutants slightly varies between the different test suites as the maximum number of killed mutants is 263 while the minimum is 260 mutants. However, the paths lengths have more variations. As shown in Fig. 8, the general trend line does not support the hypothesis. When

comparing the mutation score across all the test suites, it is observed that the general trend is a very slight increase in mutation score when the mean test paths length increases. However, the mutation score is at a peak around the median of the test paths mean lengths.

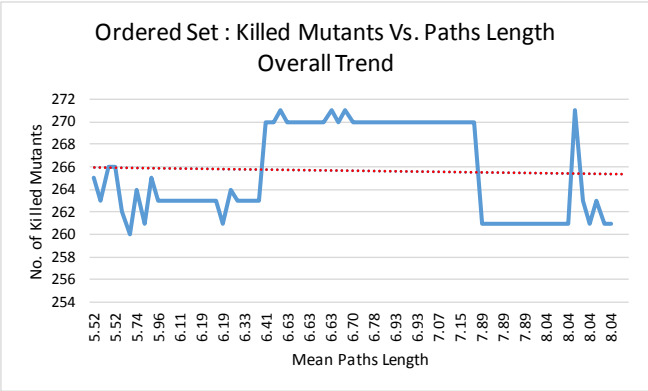


Fig. 8. Ordered Set Overall Trend Killed Mutants Vs. Mean Paths Length.

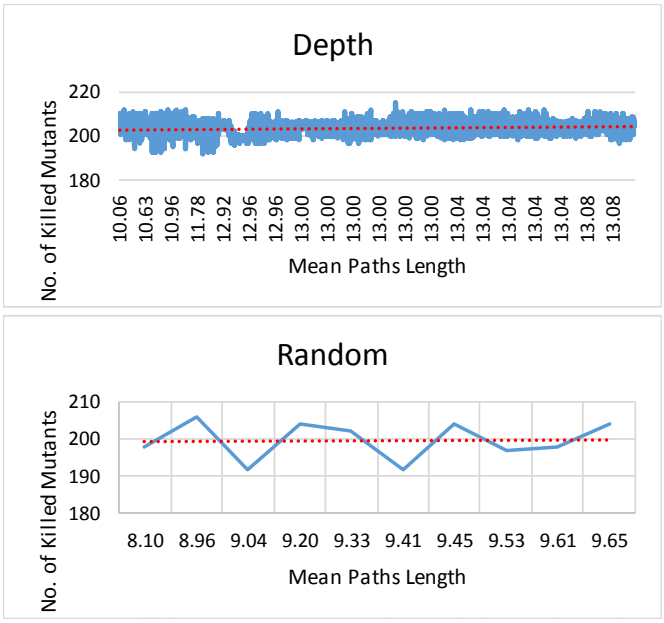
4) VCR

The VCR is the largest case study, and therefore more interesting results can be expected. The MPL across all VCR BFS test suites is identical, while the number of killed mutants varies between 150 and 206. This refutes the suggested hypothesis strongly as the mutation score varies significantly by more than 27%. For the random algorithm, the number of killed mutants varies from 192 to 206. The relationship between the mutation score and MPL is shown in Fig. 9 (a). The trend line is increasing. For the DFS test suites, the general trend is a very slow increase in mutation score relative to the MPL. Again, the peak mutation scores occur around the median of the MPL. Fig. 9 (b) shows the general trend across all test suites produced using the three different traversal algorithms. The mutation score increases in proportion to the test paths mean.

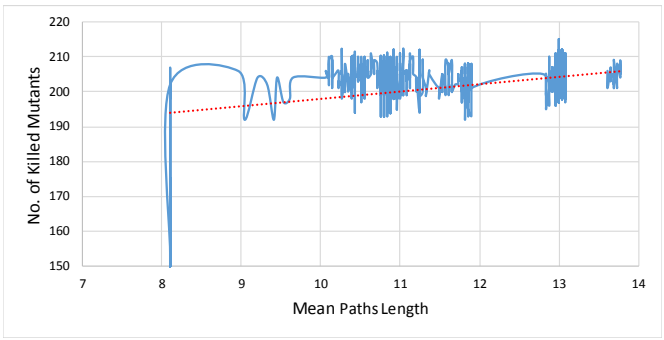
B. Hypothesis #2: Balancing the Tree Affects Mutation Score

Based on the observation pointed out in section VIII.A that high mutation score test suites have MPL close to the median paths length, we suggest the hypothesis that balanced trees (test suites) achieve better mutation score. We mean that test suites that include test cases with similar length are a better choice if one wishes to increase the mutation score. However, it has been already shown that the length affects the mutation score positively in many cases. Otherwise, balanced BFS test suites would have been the most effective. Consequently, having relatively balanced trees with lengthiest paths can achieve best results. To translate this into measurable criteria, the following steps are followed:

- Calculating the mean length of the paths in each test suite, which is already done in section VIII.A.
- Calculating the standard deviation of the paths lengths in each test suite. A tree with low standard deviation is more balanced.



(a) VCR: Killed Mutants Vs. Mean Paths Length for Random and DFS Traversals.



(b) VCR: Overall Trend for Killed Mutants Vs. Mean Paths Length.

Fig. 9. Hypothesis #1—Results for VCR

- Calculate $\frac{MPL}{Paths\ Lengths\ Standard\ Deviation\ (PLSD)}$ and look for the test suite that maximizes this ratio. This test suite has the most balanced tree with lengthiest paths.

We plotted the calculated ratio for the ordered set and the VCR against the mutation score as for smaller case studies the mutation scores has minor variations, if any.

Fig. 10 (a) and (b) show the result for ordered set and VCR, respectively. Although the results do not show that the mutation score is proportional to the ratio as suggested above, there is some interesting consistent pattern of high mutation scores (as indicated with the red dots in Fig. 10 (b)) that is especially evident in the VCR case study with the larger number of test suites. The pattern shows a tangent wave around the average mutation score value horizontal line. In other words, the mutations score constantly increases with the increase in the MPL to PLSD ratio, and then suddenly drops again. Then the mutation score goes through the same cycle repeatedly as the MPL to PLSD ratio increases. Whether the ordered set pattern appearing in Fig. 10 (a) is a snapshot of the

same behavior if we increase the number of test suites or not, is a question that needs further investigation.

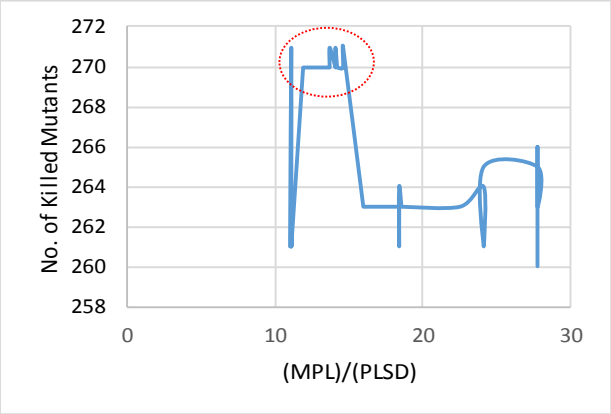
IX. Threats to Validity

Wohlin et al. [35] classify validity threats in software engineering into conclusion, internal, construct and external threats. The conclusion threat focuses on the statistical significance of the results. Despite the large number of test suites we experiment on, we still recommend applying the same experiments conducted on the study on more case studies of size closer to the VCR and the ordered set presented. Although we obtained large numbers of test suites for these case studies, we did not do any statistical analysis, which we plan to do in future work. In our experimentation, internal threats to validity are handled by avoiding variation in factors other than the factor under test. For instance, we verified that all test suites having common paths use the same inputs to the same functions. This is especially important with the ordered set since events require integer input values. For construct validity that focuses on relating the theory behind the experiments to the observations [12], Major mutation seeding tool is chosen with this concern in mind as explained in section VI. Assuming that a mutant that is covered and alive with all our test suites is an equivalent mutant, is a minor construct validity threat. However, exercising and comparing large numbers of test suites make this assumption reasonable since what affects one test suite should equally affect the others for a case study system.

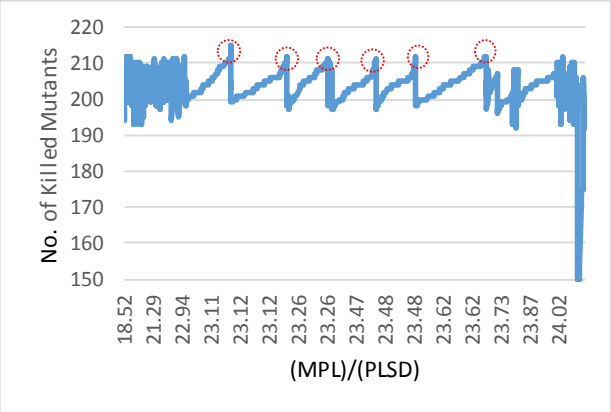
The fourth threat to validity is the external validity threat that is concerned with whether the results can be generalized outside the scope of this study. We choose case studies that are of varied sizes and characteristics to make sure they are representative of a large number of FSMs that are used for testing purposes. This is to overcome the external validity threat. However, one must be conscious in generalizing the results without experimenting with more case studies. In our results analysis, we are specific about which case studies contribute to results generalization.

X. Conclusion

In the empirical study we presented, we attempt to answer four research questions concerning FSM-based testing using transition trees and round-trip paths testing criteria. Four case studies of various sizes and characteristics are used to conduct empirical studies to answer the four research questions. The first research question to answer is whether exercising the round-trip paths in a complete manner reveals more faults than exercising those round-trip paths in a piecewise manner using transition trees. Results of the experiments on two of our case studies show that the answer to this question is yes. In one case, one fault is revealed by the round-trip paths only, while in another case study the complete round trip paths test suite reveals twenty-eight faults that are not revealed by any other test suite. The second question is whether using a certain tree traversal technique (BFS, DFS, and random) affects the mutation score of the resulting test suites. The answer to this question is yes as two of our case studies show a higher mutation score on average for DFS test suites. The third research question is whether the mutation score differs among



(a) Ordered Set



(b) VCR

Fig. 10. Number of Killed Mutants Vs. (MPL/PLSD) Ratio.

the test suites generated using the same traversal algorithm. We concluded that it does differ. The fourth and final question is whether there is a general characteristic that could be derived about the test suites with the highest mutation scores.

A couple of hypotheses are proposed based on our experimental observations. The first hypothesis is that the average length of the test cases in a test suite is proportional to the mutation score. This hypothesis is proven not to be always true. However, there appear to be a cluster of high mutation scores around the median mean paths length of tests in a test suite. This observation leads to the second hypothesis, which states that a high mutation score is achieved by the lengthiest trees with lower variance between the different test cases. Although in this research it is not proven that this hypothesis is valid, an interesting pattern is observed in the relation between the suggested ratio and the mutation score. We recommend further investigating this hypothesis using more case studies. Future work should also study relations, with our case studies (test suites, mutants), between various measures of diversity among test cases and mutation score, similarly to what others have done [16].

REFERENCES

[1] P. Amman and J. Offutt, *Introduction to Software Testing*. 2008.

[2] J. H. Andrews, L. C. Briand, and Y. Labiche, "Is mutation an appropriate tool for testing experiments?," *Proceedings. 27th Int. Conf. Softw. Eng. 2005. ICSE 2005.*, pp. 402–411, 2005.

[3] J. H. Andrews, L. C. Briand, Y. Labiche, and A. S. Namin, "Using mutation analysis for assessing and comparing testing coverage criteria," *IEEE Trans. Softw. Eng.*, vol. 32, no. 8, pp. 608–624, 2006.

[4] J. H. Andrews, A. Groce, M. Weston, and R. Xu, "Random Test Run Length and Effectiveness †," pp. 19–28, 2008.

[5] A. Arcuri, "A Theoretical and Empirical Analysis of the Role of Test Sequence Length in Software Testing for Structural Coverage," vol. 38, no. 3, pp. 497–519, 2012.

[6] B. Beizer, *Software Testing Techniques*, Second., vol. 2, no. 10. New York, 2003.

[7] E. Berard, *Testing Object-oriented Software (Abstract)*, vol. 4, no. 2. 1992.

[8] R. V. Binder, *Testing Object-Oriented Systems: Models, Patterns, and Tools*. 1999.

[9] L. C. Briand, Y. Labiche, and Y. Wang, "Using simulation to empirically investigate test coverage criteria based on statechart," *Proceedings. 26th Int. Conf. Softw. Eng.*, pp. 86–95, 2004.

[10] T. S. Chow, "Testing Software Design Modeled by Finite-State Machines," *IEEE Trans. Softw. Eng.*, vol. SE-4, no. 3, pp. 178–187, 1978.

[11] H. Do, S. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact," *Empir. Softw. Eng.*, vol. 10, no. 4, pp. 405–435, 2005.

[12] R. Feldt and A. Magazinius, "Validity Threats in Empirical Software Engineering Research - An Initial Survey.," *Proc. Int'l Conf. Softw. Eng. Knowl. Eng.*, no. August, pp. 374–379, 2010.

[13] P. Frankl and S. Weiss, "An experimental comparison of the effectiveness of branch testing and data flow testing," *Software Engineering, IEEE …*. 1993.

[14] G. Fraser, A. Gargantini, and M. Mat, "Experiments on the Test Case Length in Specification Based Test Case Generation," pp. 18–26, 2009.

[15] H. Gomaa, *Designing Concurrent, Distributed, and Real time applications with uml*. Boston: Addison Wesley, 2000.

[16] H. Hemmati, A. Arcuri, and L. Briand, "Achieving scalable model-based testing through test case diversity," *ACM Trans. Softw. Eng. Methodol.*, vol. 22, no. 1, pp. 1–42, 2013.

[17] N. E. Holt, L. C. Briand, and R. Torkar, "Empirical evaluations on the cost-effectiveness of state-based testing: An industrial case study," *Inf. Softw. Technol.*, vol. 56, no. 8, pp. 890–910, 2014.

[18] N. E. Holt, R. Torkar, L. Briand, and K. Hansen, "State-based testing: Industrial evaluation of the cost-effectiveness of round-trip path and sneak-path strategies," *Proc. - Int. Symp. Softw. Reliab. Eng. ISSRE*, pp. 321–330, 2012.

[19] R. Just, M. D. Ernst, and G. Fraser, "Efficient Mutation Analysis by Propagating and Partitioning Infected Execution States," *Proc. 2014 Int. Symp. Softw. Test. Anal.*, pp. 315–326, 2014.

[20] R. Just, D. Jalali, L. Inozemtseva, M. D. Ernst, R. Holmes, and G. Fraser, "Are mutants a valid substitute for real faults in software testing?," *Proc. 22nd ACM SIGSOFT Int. Symp. Found. Softw. Eng. - FSE 2014*, pp. 654–665, 2014.

[21] R. Just, F. Schweiggert, and G. M. Kapfhammer, "MAJOR: An efficient and extensible tool for mutation analysis in a Java compiler," *2011 26th IEEE/ACM Int. Conf. Autom. Softw. Eng. ASE 2011, Proc.*, pp. 612–615, 2011.

[22] H. Khalil and Y. Labiche, "State-based Tests Suites Automatic Generation Tool (STAGE-1)," in *IEEE Computer Society International Conference on Computers, Software & Applications.*, 2017, pp. 1–7.

[23] H. Khalil and Y. Labiche, "On FSM-Based Testing An Emperical Study: Complete Round-Trip Versus Transition Trees," in *Int. Symp. Softw. Reliab. Eng. ISSRE*, 2017.

[24] M. Khalil and Y. Labiche, "On the round trip path testing strategy," *Proc. - Int. Symp. Softw. Reliab. Eng. ISSRE*, pp. 388–397, 2010.

[25] D. Lee and M. Yannakakis, "Principles and methods of testing finite state machines - A survey," *Proc. IEEE*, vol. 84, no. 8, pp. 1090–1123, 1996.

[26] Q. Lin, "Omproving State-based Coverage Criteria Using Data.pdf." p. 202, 2004.

[27] A. P. Mathur, *foundations of software testing*. 2008.

[28] S. Mouchawrab, L. C. Briand, Y. Labiche, and M. Di Penta, "Assessing, comparing, and combining state machine-based testing and structural testing: A series of experiments," *IEEE Trans. Softw. Eng.*, vol. 37, no. 2, pp. 161–187, 2011.

[29] J. von Neumann, "Automata Studies." pp. 43–98, 1956.

[30] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empir. Softw. Eng.*, vol. 14, no. 2, pp. 131–164, 2009.

[31] A. Simão and A. Petrenko, "Fault coverage-driven incremental test generation," *Comput. J.*, vol. 53, no. 9, pp. 1508–1522, 2010.

[32] E. Suez, "State Machine Compiler." [Online]. Available: http://smc.sourceforge.net/.

[33] M. Utting and B. Legeard, *Practical Model-Based Testing*, 1st ed. Morgan Kaufmann, 2006.

[34] F. Wagner, T. Wagner, P. Wolstenholme, and F. Group, *Modeling Software with Finite State Machines A Practical Approach*. 2006.

[35] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*, vol. 9783642290. 2012.