Machine Learning Techniques for Automated Software Fault Detection via Dynamic Execution Data: Empirical Evaluation Study

Rafig Almaghairbe*, Marc Roper† and Tahani Almabruk*
*Dept. Computer Sciences
University of Omar Al-Mukhtar, Derna - Bayda, Libya
Email: rafig.almaghairbe,tahani.almabruk@omu.edu.ly
†Dept. Computer and Information Sciences
University of Strathclyde, Glasgow, UK
Email: marc.roper@strath.ac.uk

***Abstract*—The biggest obstacle of automated software testing is the construction of test oracles. Today, it is possible to generate enormous amount of test cases for an arbitrary system that reach a remarkably high level of coverage, but the effectiveness of test cases is limited by the availability of test oracles that can distinguish failing executions. Previous work by the authors has explored the use of unsupervised and semi-supervised learning techniques to develop test oracles so that the correctness of software outputs and behaviours on new test cases can be predicated [1], [2], [3], and experimental results demonstrate the promise of this approach. In this paper, we present an evaluation study for test oracles based on machine-learning approaches via dynamic execution data (firstly, input/output pairs and secondly, amalgamations of input/output pairs and execution traces) by comparing their effectiveness with existing techniques from the specification mining domain (the data invariant detector Daikon [4]). The two approaches are evaluated on a range of mid-sized systems and compared in terms of their fault detection ability and false positive rate. The empirical study also discuss the major limitations and the most important properties related to the application of machine learning techniques as test oracles in practice. The study also gives a road map for further research direction in order to tackle some of discussed limitations such as accuracy and scalability. The results show that in most cases semi-supervised learning techniques performed far better as an automated test classifier than Daikon (especially in the case that input/output pairs were augmented with their execution traces). However, there is one system for which our strategy struggles and Daikon performed far better. Furthermore, unsupervised learning techniques performed on a par when compared with Daikon in several cases particularly when input/output pairs were used together with execution traces.**

I. Introduction

Software testing is a well-established approach for ensuring software quality and reliability but it is also an expensive and time consuming process. Software developers and testers have claimed that software testing activities take more than 50% of software development time [5], [6]. To reduce testing costs, researchers have devoted considerable effort to developing automated tools to generate input data for an arbitrary system and achieve a high level of code coverage (e.g. EvoSuite [7]). However, a formal specification is not always available for the system under test and test outputs hence have to be checked manually to determine its correctness or otherwise. As a result, lots of human effort is needed if there is a large set of test cases. Therefore, some other strategy for building an oracle (an automated mechanism for judging the (in)correctness of an output associated with an input) needs to be developed to cut down cost and save time.

Previous work by the authors has explored the use of machine learning techniques to support the automatic classification of test outcomes as either passing or failing, and thereby providing a form of test oracle [1], [2], [3], but their relative performance, strengths and weaknesses have not been statistically analysed and compared with existing techniques. The aim of this study is to investigate and extensively evaluate these approaches to test oracle construction in terms of effectiveness when they are applied to medium-sized subject systems. The empirical evaluation in this paper can be summarised as follows: (1) statistical verification is implemented into two different sets of experimental results (in the first experiment, the input to the machine learning techniques consisted of just the test case inputs along with their associated outputs, and the second experiment extended this by adding to the input/output pairs their corresponding execution traces); (2) new results are presented that evaluate the effectiveness of our machine learning techniques by calculating the *accuracy*, *recall*, and *the false positive rate*; (3) a comparison between existing techniques from the specification mining domain (the data invariant detector Daikon [4]) and machine learning techniques is reported (Daikon was selected because it was the most effective oracle from a set of dynamic analysis techniques explored in a previous study [8]). The study is useful for testers because they need to be able to assess the features offered by these oracles, and also for the developers of oracle-based approaches to further understand the strengths and weaknesses of these different techniques and how they can be developed. In addition, as such approaches are developed, more work is needed on the measurement of oracles and their properties, and also it has been suggested that it is important for the software metrics community to consider the concept of ”oracle metrics” [9].

The remainder of this paper is organized as follows: Section II describes test oracle creation using machine learning techniques. Section III presents the main research question and explains the process of the empirical evaluation study design. Section IV demonstrates results and answer the main research question. Section V defines the main properties for test oracles based on machine learning techniques. Section VI discusses the threats to validity for this study. The authors conclude this paper with a summary and future work in Section VII.

II. Test Oracle Creation using Machine Learning Techniques

A. Test Oracle Based on Unsupervised Learning Techniques (Clustering)

Test oracles based on unsupervised learning techniques (clustering) do not require training data, and thus are most widely applicable. They make the implicit assumption that normal instances are far more frequent than anomalies in the test data. In other words, test oracles in this category assume that normal data instances (passing test results) belong to large and dense clusters, while anomalies (failing test results) either belong to small or sparse clusters. The inputs and outputs to the subject systems were used alone as inputs to an Agglomerative Hierarchical clustering algorithm in the first study, and then amalgamations of input/output pairs with the execution traces were used as inputs to clustering algorithm in the second study. In addition, Euclidean Distance and Linkage metrics were used with the clustering algorithm as the measure of (dis)similarity between two objects. The clustering algorithm used requires the tester to specify the number of cluster counts, and these are specified based on a percentage of the number of subject system test cases. Furthermore, the input data (input/output pairs along with execution traces) were encoded to make them more amenable to processing by machine learning algorithms. The final stage of this process is a manual one: the test case outputs need to be checked to see if they have passed or failed. This is done in cluster size order (starting with the smallest) and only considers clusters of less than average size. For further details on the experiments and methodology the reader is referred to our earlier paper [2].

B. Test Oracle Based on Semi-supervised Learning Techniques

Test oracles based on semi-supervised learning techniques assume that the training data available and has labelled instances for both failing and passing tests, or alternatively just passing tests alone. In other words, A small proportion of the test data (initially just input/output pairs and then input/output pairs with their corresponding execution traces) is labelled by the developers as passing or failing and the learning algorithms use this to build a classifier which is then used to label each remaining element (i.e. classify it as being either a passing or failing test). Self-training and co-training approaches were used in this paper. For further details on the experiments and methodology the reader is referred to an earlier paper [1].

C. Test Oracle Based on a Combination of Unsupervised/Semi-Supervised Learning Techniques

Unsupervised and semi-supervised learning strategies can be combined together to construct test oracles by using the outcomes of applying unsupervised learning techniques as input to the semi-supervised learning techniques. The test classification strategy consists of two phases: Firstly, unsupervised learning (clustering) is used with the aim of creating a grouping of tests where the smallest clusters contain a greater proportion of failures. Manual checking of tests then focuses on these smallest clusters first as they are more likely to contain failing test. Secondly, having checked a small proportion of the test outcomes, semi-supervised learning is then employed to use this information to label an initial small set of data and derive an automatic pass/fail classification for the remainder of tests [3]. The combined effect of these is to create a far more efficient process than just checking the outcome of every test in order: clustering creates a small subset of tests in which failures are more prevalent, and using semi-supervised learning allows the tester to focus next on those outputs considered to be failures.

Figure 1 shows the principles of using machine learning techniques to automatically cluster or classify (it depends on employed machine learning strategy) passing/failing outputs. The program under test is run on a set of inputs which will generate outputs and optional traces, and may encounter bugs in the program (the *'s). The pass/fail status of the outputs is unknown and the aim is to automatically distinguish these using machine learning techniques.

III. Experimental Evaluation

In this section, we describe the design of the empirical evaluation used to assess and evaluate the effectiveness of the automated test oracles generated by machine learning techniques (unsupervised and semi-supervised learning strategies) on a set of medium C/Java subject systems. This study aims to investigate one primary research question:

- Which of the automated oracle approaches is most effective in revealing new faults?

Effectiveness in this case considers various components: accuracy, recall, and the false positive rate. In addition, we investigate the statistical significance of the experimental results obtained.

As a basis for this evaluation study, the key components from experiments carried out in earlier studies [1], [2] were employed. The same subject systems from the Software Infrastructure Repository (SIR)[1], tools [2] , configuration, parameters and environment were used. The main components of the experiments are: a set of programs with known failures (these are seeded faults but are pre-defined as part of the system distribution so are outside the control of the study authors),

[1](https://sir.csc.ncsu.edu/portal/index.php)

[2]A collective classification package release 2015.2.27 (https://github.com/fracpete/collective-classification-weka-package/releases) for semi-supervised learning in WEKA release 3-6-12 (http://www.cs.waikato.ac.nz/ml/weka/downloading.html)

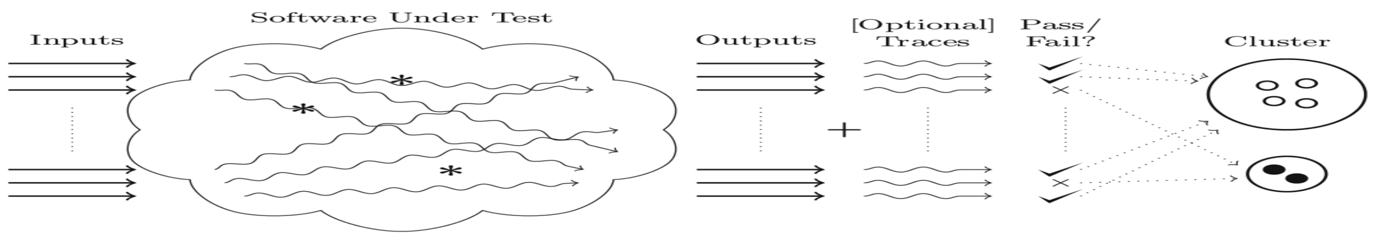

Fig. 1: Overview of Test Oracle Based on Machine Learning Strategy.

TABLE I: Characteristics of subject systems used in the study

Name	Language	Size	Test Size	Fault Types	Failure Rate	Description
NanoXML version 1	Java	7646 LOC	214	7 Seeded faults	37%	XML parser system
NanoXML version 2	Java	7646 LOC	214	7 Seeded faults	33%	XML parser system
NanoXML version 3	Java	7646 LOC	216	7 Seeded faults	31%	XML parser system
NanoXML version 5	Java	7646 LOC	216	8 Seeded faults	39%	XML parser system
Seina version 2	Java	6035 LOC	567	1 Seeded faults	17%	An event notification system
Sed version 5	C	11148 LOC	370	4 Seeded faults	18%	A stream editor

a set of test inputs for each subject system (these are also already defined and come bundled with the system), a way to determine whether an execution of each test was successful or not (pass/fail), and a mechanism for recording the execution trace taken through the subject system by each test. Details of the systems used on this study are summarised in Table I.

A. Statistical Test for Test Oracles Based on Unsupervised Learning Techniques (Clustering)

The experimental hypothesis for test oracles based on unsupervised learning techniques can be formulated in the following way: *"Normal data instances belong to large and dense clusters, while anomalies either belong to small or sparse cluster"* [10]. The hypothesis test will be employed on several experimental data for all subject programs used in this paper to test and see the impact of clustering approach. Note that, if the clustering approach has no impact then the failures will be evenly distributed throughout the clusters irrespective of their size. A null hypothesis, alternative hypothesis and significance level are stated as follows:

- The Null hypothesis (H_0) $P <= (FN/TZ)$ (The proportion of failures found in small clusters is less than or equal to FN/TZ). Where FN is all the failures in test suite, and TZ is the size of the test suite.
- The Alternative hypothesis (H_1) $P > (FN/TZ)$ (The proportion of failures found in small cluster is more than FN/TZ).
- The significance level is 0.05, and Z-Test method with a right one-tailed test is selected (the relatively large sample size makes the Z-Test an appropriate choice in this case), as defined by the following equation:

$$Z = (P - P_0)/\sqrt{P_0(1 - P_0)/N} \tag{1}$$

Where P is the proportion of failures found in small clusters, and P_0 is the proportion of all failures in the test suite.

B. Statistical Test for Test Oracles Based Semi-supervised Learning Techniques

The experimental hypothesis for test oracles based on semi-supervised learning techniques can be formulated in the following way: *"Test oracles based on semi-supervised learning techniques are able to accurately classify a significant majority of unlabelled (or unseen) data"*. The binomial test will be used to test the hypothesis on the experimental results for all subject programs used in this paper. A null hypothesis, alternative hypothesis, equation parameters and significance level are stated as follows:

- The Null hypothesis (H_0) $P <= 0.5$. Where P is the classification probability compared against random chance classification probability obtained by (0.5). In other words, it is compared against a random oracle where a random oracle is a classifier which classifies data by random chance.
- The Alternative hypothesis (H_1) $P > 0.5$.
- N is the number of errors for the classifier (FP+FN) under a particular labelled data size, K is the number of tests which represent the number of unlabelled data that the classifier attempts to classify.
- The significance level is 0.05, and Z-Test method with a right one tailed test is used, as defined by the following equation:

$$Z = ((K - NP)0.5)/\sqrt{NPq} \tag{2}$$

Where q is the classification accuracy of random oracle which is equal to 0.5.

C. Comparison with Daikon

To try and provide some meaningful comparison regarding the effectiveness of the approach presented in this paper, a comparison with Daikon [4] is also presented. Daikon is a popular tool in the specification mining area that can also be used as an automated test oracle. Daikon is dynamic analyser that is able to infer likely program invariants from the synthesis of program properties (e.g. key variables and relationships) observed over several program traces. An invariant is a property that holds at a certain point or points in a program; these are often used in assert statements, documentation and formal specifications. Daikon instruments and runs a program, observes the values that the program computes, and then reports properties that were true over the observed executions.

To build the initial set of assertions Daikon was run on the *non*-faulty version of each system using the same supplied set of test cases. Following this, Daikon was run on the various versions of the program containing the seeded faults (again using the supplied test cases) and used to establish whether there were any violations of the initially established assertions. To confirm that Daikon detects a seeded fault (true positive), we manually inspected the output reports produced by Daikon to find if there is any direct link between the reports (the violated invariants) and the seeded faults (information about the seeded faults can be found for each subject program in SIR). A true positive is noted if the trained oracle (Daikon assertions) reports an alarm that is verified to point to a corresponding faults. A false positive is recorded if the trained oracle reports an alarm which does not relate to any of the corresponding faults. If the trained oracle does not report any alarm in relation to a seeded fault then this can be considered as false negative.

It must be stressed that Daikon assumes that the system under test has fault-free version on which to train the oracle - something which is difficult to obtain in the reality. In contrast our approach makes no such assumption about a fault-free version.

IV. Experimental Results and Discussion

A. Test Oracle Creation Using Unsupervised Learning Techniques

1) Study 1: Unsupervised Learning Techniques (Clustering Input/Output Pairs) Versus Daikon: In this comparative study, we replicate the best results achieved in our earlier paper [2] for test oracles based on the Agglomerative Hierarchical clustering algorithm with the same parameters (such as the number of clusters), and just using input/output pairs of subject programs. The results of this are summarised in Table II which shows in column 1 ("Test Oracles") either the type of Linkage metric used with the Agglomerative Hierarchical clustering algorithm (Single, Average, or Complete) or if the result comes from the Daikon study, in column 2 the number of clusters specified as parameter by the developer – CC (expressed as a percentage of number of subject program test cases) and the average size of small (less than average size) clusters – CS (again expressed as a percentage of number of subject program test cases)[3], in column 3 details of confusion matrix (CM), and finally in column 4 the evaluation metrics (FPR: false positive rate, REC: recall rate, ACC: Accuracy rate). The confusion matrix is expressed in term of true positives (TP: a failing test result that appears in a small cluster), true negative (TN: a passing test result that appears in large cluster), false positive (FP: a passing test result that appears in small cluster) and false negative (FN: a failing test result that appears in a large cluster). The false positive rate is defined as the ratio of incorrectly detected failures to the number of all non-failures. Recall is the ratio of correctly detected failures to the number of true failures. Accuracy is defined as the ratio of all correct classifications to the number of all classifications that have been performed – in simple terms, how well the clustering algorithm is doing at separating the failing tests into the smaller clusters and the passing ones into the larger clusters. The best results in terms of these last three metrics are highlighted in the table in bold.

The experimental results for NanoXML version 1 and 5 show that Agglomerative Hierarchical clustering algorithm with Single and Average linkage metrics performed better than Daikon. The false positive rate (FPR) was between 0.16 to 0.20 and the failure detection rate (REC) was between 56% to 61%, and the accuracy rate (ACC) reflects this observation. However, for NanoXML version 2 and 3 Daikon slightly performed better than the Agglomerative Hierarchical clustering algorithm with all linkage metrics. In addition, the clustering algorithm with all linkage metrics was not able to outperformed Daikon for Siena and Sed, although for Siena it achieved a higher failure detection rate but slightly lower accuracy rate. It's worth remembering that the information used to construct the test oracle for Daikon (execution traces) contains more details compared to the information used to build a test oracle for the clustering algorithm (only input/output pairs). In general, the overall performance of Agglomerative Hierarchical clustering as a test oracle is acceptable compared to Daikon (the average false positive rate is 19% on all systems and 17% for Daikon, and the average failure detection rate is 61% on all systems and 64% for Daikon).

2) Study 2: Unsupervised Learning Techniques (Clustering Input/Output Pairs and Execution Traces) Versus Daikon: In this comparative study, we replicate the best results achieved in our earlier paper [2] for test oracles based on the Agglomerative Hierarchical clustering algorithm using input/output pairs with their corresponding execution trace of subject programs, and again with the same parameters (such as the number of clusters). The results of this are summarised in Table III. The tables columns contain the same type of information as Table II

The results for Nanoxml versions 1, 2 and 5 show that Agglomerative Hierarchical clustering with various Linkage metrics outperformed Daikon. The false positive rate (FPR) was between 0 to 0.10 and the failure detection (REC) was between 64% to 78%, and this also can be observed in the accuracy rate (ACC). Only for Nanoxml version 3 did Daikon perform slightly better. For Siena the results show that Agglomerative Hierarchical clustering with Complete metric outperformed Daikon with similar false positive rate but higher

[3]So for the first entry in the table, NanoXML has 214 tests which means that for the Single Linkage case there were 32 clusters (15% of 214) and the average size of the small clusters was 6 (3% of 214)

TABLE II: Unsupervised Learning Techniques (Clustering Input/Output Pairs) Versus Daikon

Nanoxml Version 1			
Test Oracles	CC, CS	CM (TP,FP,FN,TN)	FPR, REC, ACC
Single	(15%, 3%)	(46, 21, 35, 105)	**(0.16, 0.56, 0.72)**
Average	(5%, 10%)	(46, 26, 35, 100)	(0.20, 0.56, 0.70)
Complete	(15%, 3.12%)	(48, 41, 33, 85)	(0.32, 0.59, 0.64)
Daikon	//	(26, 39, 44, 40)	(0.49, 0.37, 0.44)
Nanoxml Version 2			
Test Oracles	CC, CS	CM (TP,FP,FN,TN)	FPR, REC, ACC
Single	(10%, 3.5%)	(45, 10, 26, 126)	(0.07, 0.63, 0.82)
Average	(5%, 10%)	(45, 26, 26, 110)	(0.19, 0.63, 0.74)
Complete	(25%, 2.25%)	(46, 45, 25, 91)	(0.33, 0.64, 0.66)
Daikon	//	(45, 11, 15, 78)	**(0.12, 0.75, 0.82)**
Nanoxml Version 3			
Test Oracles	CC, CS	CM (TP,FP,FN,TN)	FPR, REC, ACC
Single	(20%, 2.5%)	(51, 14, 19, 122)	(0.10, 0.72, 0.83)
Average	(15%, 3.25%)	(58, 15, 12, 122)	(0.02, 0.82, 0.86)
Complete	(15%, 3.12%)	(59, 15, 11, 122)	(0.10, 0.84, 0.87)
Daikon	//	(63, 14, 6, 86)	**(0.14, 0.91, 0.88)**
Nanoxml Version 5			
Test Oracles	CC, CS	CM (TP,FP,FN,TN)	FPR, REC, ACC
Single	(15%, 3%)	(40, 28, 25, 114)	**(0.19, 0.61, 0.74)**
Average	(10%, 6.25%)	(40, 29, 25, 113)	**(0.20, 0.61, 0.74)**
Complete	(15%, 3.12%)	(36, 61, 29, 81)	(0.42, 0.55, 0.56)
Daikon	//	(35, 20, 30, 74)	(0.21, 0.53, 0.68)
Siena Version 2			
Test Oracles	CC, CS	CM (TP,FP,FN,TN)	FPR, REC, ACC
Single	(20%, 0.79%)	(61, 42, 23, 368)	(0.10, 0.72, 0.86)
Average	(25%, 0.60%)	(63, 55, 21, 355)	(0.13, 0.75, 0.84)
Complete	(20%, 0.79%)	(61, 18, 23, 392)	(0.10, 0.72, 0.86)
Daikon	//	(60, 20, 24, 390)	**(0.04, 0.71, 0.91)**
Sed Version 5			
Test Oracles	CC, CS	CM (TP,FP,FN,TN)	FPR, REC, ACC
Single	(25%, 1%)	(26, 95, 46, 195)	(0.32, 0.36, 0.61)
Average	(25%, 0.8%)	(21, 65, 51, 225)	(0.22, 0.29, 0.67)
Complete	(25%, 1%)	(30, 85, 42, 205)	(0.29, 0.41, 0.64)
Daikon	//	(40, 20, 26, 276)	**(0.06, 0.60, 0.87)**

TABLE III: Unsupervised Learning Techniques (Clustering Input/Output Pairs and Execution Traces) Versus Daikon

Nanoxml Version 1			
Test Oracles	CC, CS	CM (TP,FP,FN,TN)	FPR, REC, ACC
Single	(15%, 4.47%)	(45, 8, 25, 71)	**(0.10, 0.64, 0.77)**
Average	(15%, 3.43%)	(45, 8, 25, 71)	**(0.10, 0.64, 0.77)**
Complete	(25%, 3%)	(41, 31, 29, 48)	(0.39, 0.58, 0.59)
Daikon	//	(26, 39, 44, 40)	(0.49, 0.37, 0.44)
Nanoxml Version 2			
Test Oracles	CC, CS	CM (TP,FP,FN,TN)	FPR, REC, ACC
Single	(15%, 5.78%)	(47, 9, 13, 80)	**(0.10, 0.78, 0.85)**
Average	(15%, 4.43%)	(47, 9, 13, 80)	**(0.10, 0.78, 0.85)**
Complete	(25%, 3%)	(42, 31, 18, 58)	(0.34, 0.70, 0.67)
Daikon	//	(45, 11, 15, 78)	(0.12, 0.75, 0.82)
Nanoxml Version 3			
Test Oracles	CC, CS	CM (TP,FP,FN,TN)	FPR, REC, ACC
Single	(15%, 4.37%)	(57, 9, 12, 91)	(0.09, 0.82, 0.87)
Average	(20%, 3%)	(52, 26, 17, 74)	(0.04, 0.75, 0.74)
Complete	(15%, 3.92%)	(59, 6, 10, 94)	(0.06, 0.85, 0.90)
Daikon	//	(63, 14, 6, 86)	**(0.14, 0.91, 0.88)**
Nanoxml Version 5			
Test Oracles	CC, CS	CM (TP,FP,FN,TN)	FPR, REC, ACC
Single	(15%, 4.37%)	(39, 20, 26, 74)	(0.21, 0.60, 0.71)
Average	(15%, 4.41%)	(39, 24, 26, 70)	(0.25, 0.60, 0.71)
Complete	(10%, 6.50%)	(46, 0, 19, 94)	**(0, 0.70, 0.88)**
Daikon	//	(35, 20, 30, 74)	(0.21, 0.53, 0.68)
Siena Version 2			
Test Oracles	CC, CS	CM (TP,FP,FN,TN)	FPR, REC, ACC
Single	(20%, 0.79%)	(63, 42, 21, 368)	(0.10, 0.75, 0.87)
Average	(10%, 1.98%)	(63, 45, 21, 365)	(0.10, 0.75, 0.86)
Complete	(5%, 4.04%)	(84, 18, 0, 392)	**(0.04, 1, 0.96)**
Daikon	//	(60, 20, 24, 390)	(0.04, 0.71, 0.91)
Sed Version 5			
Test Oracles	CC, CS	CM (TP,FP,FN,TN)	FPR, REC, ACC
Single	(20%, 1.20%)	(23, 71, 43, 225)	(0.23, 0.34, 0.56)
Average	(25%, 1%)	(26, 104, 40, 192)	(0.35, 0.39, 0.60)
Complete	(25%, 1%)	(25, 105, 41, 191)	(0.35, 0.37, 0.59)
Daikon	//	(40, 20, 26, 276)	**(0.06, 0.60, 0.87)**

failure detection rate and accuracy rate. In contrast Daikon substantially outperformed the clustering algorithm approach for the Sed subject system.

After comparing the experimental results in Table II and Table III, it can be noticed that a test oracle based on clustering using input/output pairs with execution traces performed better than a test oracle based on clustering using input/output pairs only.

As part of the evaluation in this paper, a statistical test is applied to the experimental results. The results of this are summarised in Table IV which shows the selected test number for a specific system and its version (Test 1 to Test 6 are selected from Table II as a sample for both studies), the oracle type, failures found - proportion of failures found on the smallest clusters, population size - the size of all test data on the smallest clusters (true positive and false positive), *z-test* score, and *p-value* score.

The null hypothesis (see section III-A) is rejected in all cases. It can be observed that the experimental results are statistically significant in all cases where $p-value$ is lower than the significance level (0.05). This indicates that the clustering of failures into the smaller clusters is a consequence of the technique applied and better than random.

B. Test Oracle Creation Using Semi-Supervised Learning Techniques

1) Study 1: Semi-Supervised Learning Techniques (Classifying Input/Output Pairs) Versus Daikon: Test oracles based on semi-supervised learning techniques can be constructed in two different scenarios. In the first scenario, classifiers were trained on a small set of labelled instances of the normal behaviour class (in this case, passing executions) and a few instances of the abnormal behaviour class (failing executions). The rest of data were unlabelled instances of input/output pairs which the classifiers iteratively categorised during the learning process. In the second scenario, classifiers were trained on a small set of labelled instances for normal behaviour class (passing executions) alone, with the remaining data being unlabelled instances (as unknown class for the classifier during the training process).

In this comparative study, we replicate the best results achieved in [1] for test oracles based on self-training (using Naïve Bayes classifier with Expectation-Maximisation clustering algorithm) and co-training (using Naïve Bayes classifier) with the same parameters such as the size of labelled data (the same labelled data were also selected), and using one set of features (input/output pairs) to build the classifiers. The results

TABLE IV: Statistical Test Results for Test Oracles Based on Unsupervised Learning Techniques

Test Number	Test Oracles	Failures Found	Population Size	*z-test*	*p-value*
Test 1 for NanoXML version 1	Single	56%	67	2.4419	< 0.00734
Test 2 for NanoXML version 2	Single	63%	55	3.9008	0.00005
Test 3 for NanoXML version 3	Average	82%	73	4.6684	0.00001
Test 4 for NanoXML version 5	Average	61%	69	4.2723	0.00001
Test 5 for Seina version 2	Complete	72%	79	10.5417	< 0.00001
Test 6 for Sed version 5	Complete	41%	121	4.885	< 0.00001

of this are summarised in Table V. The tables columns contain the same type of information as Table II and Table III with the exception of the second column (LS) which indicates the size of labelled data set (based on a percentage of the number of subject program test cases).

The experimental results for semi-supervised learning techniques using input/output pairs under scenario 1 (a proportion of both normal and abnormal data are labelled) for NanoXML show that self-training outperformed Daikon on version 1 and 5. The false positive rate (FPR) was 0.12 and failure detection rate (REC) was between 0.76 to 0.80 on both version. On the other hand, Daikon performed slightly better on NanoXML versions 2 and 3 where the failure detection rate is slightly higher (13% higher on average for both versions), however, self-training classifier has lower false positive rate (7% lower on average for both version). The experimental results for Siena and Sed show that Daikon outperformed all semi-supervised learning methods (a test oracle based on Daikon achieved a lower false positive rate and higher failure detection rate for both systems).

The results for semi-supervised learning techniques using input/output pairs under scenario 2 (labelling subsets of only normal data) are omitted because none of the approaches performed well enough. The majority of the time the trained oracles (classifiers) were only able to classify all passing (normal) execution data correctly but misclassified all failing (abnormal) execution data, labelling it as normal data instead.

2) Study 2: Semi-Supervised Learning Techniques (Classifying Input/Output Pairs and Execution Traces) Versus Daikon: In this comparative study, we replicate the best results achieved in [1] for test oracles based on self-training (using Naïve Bayes classifier with Expectation-Maximisation clustering algorithm) and co-training (using Naïve Bayes classifier) with the same parameters such as the size of labelled data (the same labelled data were also selected), and using two sets of features (input/output pairs and execution traces) to build the classifiers. The results of this are summarised in Table VI and Table VII. The tables columns contain the same type of information as Table V.

Table VI presents the results of using semi-supervised learning techniques on input/output pairs augmented with their execution traces for all versions of subject systems under scenario 1 (both normal and abnormal data are labelled). Again the best performing results are indicated in bold. From the experimental results, it can be observed that practically all the semi-supervised learning approaches outperformed Daikon for

TABLE V: Semi-supervised Learning Techniques (Classifying Input/Output Pairs) Versus Daikon - Scenario 1

Nanoxml Version 1			
Test Oracles	LS	CM (TP,FP,FN,TN)	FPR, REC, ACC
Self	40%	(65, 16, 16, 110)	**(0.12, 0.80, 0.84)**
Co-Naive	40%	(13, 0, 68, 126)	(0, 0.16, 0.67)
Daikon	//	(26, 39, 44, 40)	(0.49, 0.37, 0.44)
Nanoxml Version 2			
Test Oracles	LS	CM (TP,FP,FN,TN)	FPR, REC, ACC
Self	30%	(45, 9, 26, 127)	**(0.06, 0.63, 0.83)**
Co-Naive	10%	(13, 7, 58, 129)	(0.05, 0.18, 0.68)
Daikon	//	(45, 11, 15, 78)	**(0.12, 0.75, 0.82)**
Nanoxml Version 3			
Test Oracles	LS	CM (TP,FP,FN,TN)	FPR, REC, ACC
Self	40%	(55, 9, 15, 128)	**(0.06, 0.78, 0.88)**
Co-Naive	10%	(12, 7, 58, 130)	(0.05, 0.17, 0.68)
Daikon	//	(63, 14, 6, 86)	**(0.14, 0.91, 0.88)**
Nanoxml Version 5			
Test Oracles	LS	CM (TP,FP,FN,TN)	FPR, REC, ACC
Self	50%	(50, 18, 15, 124)	**(0.12, 0.76, 0.84)**
Co-Naive	10%	(14, 0, 51, 142)	(0, 0.21, 0.75)
Daikon	//	(35, 20, 30, 74)	(0.21, 0.53, 0.68)
Siena Version 2			
Test Oracles	LS	CM (TP,FP,FN,TN)	FPR, REC, ACC
Self	10%	(51, 209, 33, 201)	(0.50, 0.60, 0.51)
Co-Naive	10%	(13, 98, 71, 312)	(0.23, 0.15, 0.23)
Daikon	//	(60, 20, 24, 390)	**(0.04, 0.71, 0.91)**
Sed Version 5			
Test Oracles	LS	CM (TP,FP,FN,TN)	FPR, REC, ACC
Self	40%	(24, 4, 42, 292)	(0.01, 0.36, 0.87)
Co-Naive	40%	(24, 4, 42, 292)	(0.01, 0.36, 0.87)
Daikon	//	(40, 20, 26, 276)	**(0.06, 0.60, 0.87)**

all versions of Nanoxml and the Siena subject systems. The false positive rate (FPR) was between 0 to 0.05 and the failures detection (REC) was between 94% to 100%, and this also can be observed in the accuracy rate (ACC). This is quite a remarkable result: for some of the systems it is possible to build a completely accuracte classifier after training it on just 10% of the samples - i.e just over 20 test cases. On the other hand, semi-supervised learning techniques for Sed subject system did not perform well in comparison to Daikon as they have a high false positive rate and lower accuracy rate although more failures (true positives) are detected. The test data (input/output pairs and execution traces) for the Sed subject system were examined and it was observed that Sed has the most fragmented input-output combination which combined with its 295 distinct traces gives a 341 distinct input-output-trace combinations (in comparison to 120 for NanoXML and 104 for Siena). For cases such as this it is clear either that other information needs to be introduced to the classifier algorithms such as execution time or summary information relating to traces (e.g. number of unique methods, nesting pattern etc.), or that the volume of data observations needs to be increased to give the machine learning algorithms a greater chance of identifying patterns (as it stands there is on average just over one test case per unique trace).

Table VII shows the results of using input/output pairs augmented with their execution traces along with scenario 2 (labelling a proportion of the normal test results only). Semi-supervised learning techniques again outperformed Daikon in all versions for Nanoxml and Siena except version 1 of NanoXML where they performed on a par (for Nanoxml version 1 Daikon has a high false positive rate and a low accuracy rate but a higher failure detection rate). Again these results are worth noting, especially as the models in this case had not seen any examples of failing test cases. However, the results for Sed show that Daikon beat the semi-supervised learning techniques (although the failure detection rate - the TP values - for self-training is slightly higher compared to Daikon).

Overall the semi-supervised learning techniques performed well in comparison to Daikon (regardless of whether the training used both normal and abnormal data labels or solely normal labels) on both the NanoXML and Siena systems. The relatively weaker performance of Daikon may be attributable to the size of the test suites - it may have been that the suites were all too small to adequately train the oracle. However, the same data sets were used for training the semi-supervised learning approaches which may be an advantage for these techniques if they are able to perform well even with a small test suite size.

For Sed by contrast our machine learning approach failed to achieve anything close to the performance obtained on other systems and was Daikon consistently performed better. Remember that Daikon has the advantage of a "clean" version of the system to use to build the assertions, but still outperformed our semi-supervised learning approach. This inconsistency of performance is something we observed in previous studies and is something we are working on to address by exploring different trace information, data encoding (of inputs, outputs and traces), distance measures and algorithms.

To evaluate the performance of semi-supervised learning techniques as test oracles, statistical tests are applied to the experimental results. The results of this are summarised in Table VIII which shows the selected test number for specific system and its version (Test 1 to Test 6 are selected from Table V as sample for both studies), the oracle type, the classifier accuracy, the number of errors - ($N = FP+FN$), the number of tests - (K) is the number of tests which represent the number of unlabelled data that the classifier attempts to classify, *z-test* score, and *p-value* score.

The null hypothesis (see section III-B) is rejected in all cases. It is observed that the results are statistically significant in all cases where *p-value* lower than the significance level (0.05). This again demonstrates that the impact of the classifier is far greater than could be achived by chance.

TABLE VI: Semi-supervised Learning Techniques (Classifying Input/Output Pairs with Execution Traces) Versus Daikon - Scenario 1

Nanoxml Version 1			
Test Oracles	LS	CM (TP,FP,FN,TN)	FPR, REC, ACC
Self	50%	(66, 4, 4, 75)	**(0.05, 0.94, 0.94)**
Co-Naive	10%	(28, 28, 42, 51)	(0.35, 0.40, 0.53)
Daikon	//	(26, 39, 44, 40)	(0.49, 0.37, 0.44)
Nanoxml Version 2			
Test Oracles	LS	CM (TP,FP,FN,TN)	FPR, REC, ACC
Self	10%	(60, 0, 0, 89)	**(0, 1, 1)**
Co-Naive	10%	(60, 0, 0, 89)	**(0, 1, 1)**
Daikon	//	(45, 11, 15, 78)	(0.12, 0.75, 0.82)
Nanoxml Version 3			
Test Oracles	LS	CM (TP,FP,FN,TN)	FPR, REC, ACC
Self	10%	(69, 0, 0, 100)	**(0, 1, 1)**
Co-Naive	10%	(69, 0, 0, 100)	**(0, 1, 1)**
Daikon	//	(63, 4, 6, 86)	(0.04, 0.91, 0.93)
Nanoxml Version 5			
Test Oracles	LS	CM (TP,FP,FN,TN)	FPR, REC, ACC
Self	10%	(65, 0, 0, 94)	**(0, 1, 1)**
Co-Naive	10%	(65, 0, 0, 94)	**(0, 1, 1)**
Daikon	//	(35, 20, 30, 74)	(0.21, 0.53, 0.68)
Siena Version 2			
Test Oracles	LS	CM (TP,FP,FN,TN)	FPR, REC, ACC
Self	10%	(84, 0, 0, 410)	**(0, 1, 1)**
Co-Naive	10%	(84, 0, 0, 410)	**(0, 1, 1)**
Daikon	//	(60, 20, 24, 390)	(0.04, 0.71, 0.91)
Sed Version 5			
Test Oracles	LS	CM (TP,FP,FN,TN)	FPR, REC, ACC
Self	30%	(57, 86, 9, 210)	(0.29, 0.86, 0.73)
Co-Naive	40%	(56, 86, 10, 210)	(0.29, 0.84, 0.73)
Daikon	//	(40, 20, 26, 276)	**(0.06, 0.60, 0.87)**

C. Combining Unsupervised/Semi-Supervised/Supervised Learning Techniques Versus Daikon

In this approach to building test oracles, developers can use the results obtained from unsupervised learning techniques (clustering) as input to a supervised learning algorithm in order to reduce the search space of required labelled training data set (labelled training data set is not always available). Developers manually check the test results on small clusters and labelled them as a passed or failed test, and then use this labelled data to train a supervised learning algorithm and to classify the remaining test results data.

In this study, we select the results obtained from applying Agglomerative Hierarchical clustering algorithm with the Single Linkage metric for Nanoxml subject system version 1 as input to Naïve Bayes classifier (Table IX), and this could be considered as a case study for this approach. After examining and labelling the small clusters (where small sized clusters equate to 4.47% of the test data - i.e. 7 instances or less) a low classification accuracy of 23% is obtained. However, the classifier successfully detects all failures (recall is higher than Daikon) but the fault detection capability comes at the price of high false positive rate (Daikon has 50% false positive rate lower). This is caused by using an imbalanced labelled training data set to train the classifier (the training data set has 45 failing tests and just 8 passing tests) which appears to be causing passing tests to be wrongly classified as failing. This imbalance is a consequence of the success

TABLE VII: Semi-supervised Learning Techniques (Classifying Input/Output Pairs with Execution Traces) Versus Daikon - Scenario 2

Nanoxml Version 1			
Test Oracles	LS	CM (TP,FP,FN,TN)	FPR, REC, ACC
Self	50%	(17, 2, 53, 77)	**(0.02, 0.24, 0.63)**
Co-Naive	50%	(0, 0, 70, 79)	(0, 0, 0.53)
Daikon	//	(26, 39, 44, 40)	(0.49, 0.37, 0.44)
Nanoxml Version 2			
Test Oracles	LS	CM (TP,FP,FN,TN)	FPR, REC, ACC
Self	40%	(60, 0, 0, 89)	**(0, 1, 1)**
Co-Naive	10%	(58, 0, 2, 89)	(0, 0.96, 0.98)
Daikon	//	(45, 11, 15, 78)	(0.12, 0.75, 0.82)
Nanoxml Version 3			
Test Oracles	LS	CM (TP,FP,FN,TN)	FPR, REC, ACC
Self	20%	(67, 0, 2, 100)	(0, 0.97, 0.98)
Co-Naive	10%	(69, 0, 0, 100)	**(0, 1, 1)**
Daikon	//	(63, 4, 6, 86)	(0.04, 0.91, 0.93)
Nanoxml Version 5			
Test Oracles	LS	CM (TP,FP,FN,TN)	FPR, REC, ACC
Self	50%	(65, 0, 0, 94)	**(0, 1, 1)**
Co-Naive	40%	(42, 0, 23, 94)	(0, 0.64, 0.85)
Daikon	//	(35, 20, 30, 74)	(0.21, 0.53, 0.68)
Siena Version 2			
Test Oracles	LS	CM (TP,FP,FN,TN)	FPR, REC, ACC
Self	10%	(84, 0, 0, 410)	**(0, 1, 1)**
Co-Naive	20%	(84, 0, 0, 410)	**(0, 1, 1)**
Daikon	//	(60, 20, 24, 390)	(0.04, 0.71, 0.91)
Sed Version 5			
Test Oracles	LS	CM (TP,FP,FN,TN)	FPR, REC, ACC
Self	50%	(45, 63, 21, 233)	(0.21, 0.68, 0.76)
Co-Naive	50%	(32, 44, 34, 252)	(0.14, 0.48, 0.78)
Daikon	//	(40, 20, 26, 276)	**(0.06, 0.60, 0.87)**

of the unsupervised learning and is something that could be addressed by re-balancing techniques (e.g. randomly sampling from large clusters on the assumption that they are unlikely to contain failures) and is also something that we need to investigate further.

We also studied the use of clustering algorithm results as input to a *semi*-supervised learning classifier where the labelled test results are combined with unlabelled test results (unknown test results) in the training phase. This could be a solution for the imbalanced training data set problem (the classifier is trained iteratively and training data set is also updated iteratively). The results for this (Table X) show that a low false positive rate is achieved but also that a lower fault detection ability is obtained which makes the balance between false positive rate and failure detection rate a challenging task for the development of automated test oracles.

V. Test Oracles Via Machine Learning Techniques And Their Properties

In terms of the application of machine learning techniques as test oracles in practice, the most important properties that test oracles need to demonstrate are scalability, fault detection ability, false positive rate and cost effectiveness. Each of those properties is explained further below:

- Scalability means the ability of any technique to handle any size of software (with corresponding increases in the volume of data). In other words, a technique has to be potentially usable at an industrial level.
- Fault detection ability refers to the effectiveness which new (unseen) faults occurring in running application are identified.
- False positive rate is the rate of false alarms reported by test oracles. This can be considered as the biggest issue with automated oracles. When such a rate is intolerably high, any problem reported by automated oracles will be deemed unreliable and ignored by developers.
- Cost effectiveness takes into consideration the effort and resources required to create an oracle in relation to its ability to reveal subtle semantic failures.

Generally, all those properties are complementary to each other and can affect the usability of any test oracle in practice. The ultimate goal of the software testing community is to find a test oracle that can be used to test any system, and is able to find all failures with a low false positive rate at an acceptable cost.

Test oracles based on unsupervised learning techniques do not require the availability of labelled data or a fault free version of the system under test to construct test oracles which make them more scalable in comparison to test oracles based on supervised/semi-supervised learning techniques and test oracles based on invariant detection in terms of the provision of labelled data (other scalability issues may arise in the application of the algorithms but these a likely to be equally applicable to all approaches). In addition, the presented approaches (unsupervised learning techniques) in this paper are less expensive to obtain in comparison to test oracles based on supervised/semi-supervised learning techniques (again as no data labelling is necessary), but can be less accurate for the same reason.

Test oracles based on semi-supervised learning techniques are more less expensive in comparison to those based on supervised learning techniques as they require a smaller set of labelled training data (as opposed to the large data set required by supervised techniques or the fault-free version employed by invariant detectors). However, oracles based on semi-supervised learning techniques have a lower accuracy in comparison to those based on supervised learning techniques (they have slightly higher false positive rate, and also slightly lower fault detection ability) which is to be expected as the training of the algorithms uses far less labelled data. Semi-supervised approaches are a classic demonstration of the cost-benefit trade-off: a larger set of labelled data is likely to yield a more accurate classifier, and while these techniques are significantly more cost-effective (and practicable) than supervised approaches, there is still work to be done in establishing the ideal ratio of labelled to unlabelled data.

VI. Threats to Validity

The clear issue concerning the external validity of this study is the generalizability of our results: the findings so far are limited to three subject programs which cannot be said to form a representative set, even though they are non-trivial real-world Java and C systems of a reasonable size containing real faults. The failure rates for all systems may also not be

TABLE VIII: Statistical Test Results for Test Oracles Based on Semi-supervised Learning Techniques

Test Number	Test Oracles	Accuracy	(K)	(N)	*z-test*	*p-value*
Test 1 for NanoXML version 1	Self	0.84	123	32	37.653436	< 0.00001
Test 2 for NanoXML version 2	Self	0.83	145	35	42.933836	< 0.00001
Test 3 for NanoXML version 3	Self	0.88	124	24	45.519684	< 0.00001
Test 4 for NanoXML version 5	Self	0.84	104	33	30.289512	< 0.00001
Test 5 for Siena version 2	Self	0.51	444	242	1.850858	< 0.032099
Test 6 for Sed version 5	Self	0.87	212	46	-8.172954	< 0.00001

TABLE IX: Unsupervised/Supervised Learning Techniques Versus Daikon for NanoXML

Nanoxml Version 1			
Test Oracles	(CC,CS,LS)	CM (TP,FP,FN,TN)	FPR, REC, ACC
Single-Naïve Bayes	15%, 4.47%, 35%	(23, 73, 0, 0)	(1, 1, 0.23)
Daikon	//	(26, 39, 44, 40)	(0.49, 0.37, 0.44)

TABLE X: Unsupervised/Semi-supervised Learning Techniques Versus Daikon for NanoXML

Nanoxml Version 1			
Test Oracles	(CC,CS,LS)	CM (TP,FP,FN,TN)	FPR, REC, ACC
Single-Naïve Bayes	15%, 4.47%, 35%	(12, 0, 11, 73)	(0, 0.52, 0.88)
Daikon	//	(26, 39, 44, 40)	(0.49, 0.37, 0.44)

representative, as may be the test cases (although these were created independently via collaboration between the SIR and the subject systems' developers).

A potential construct validity for this study lies on our use of the coding scheme for both input/output pairs and execution traces. However, for the input/output pairs, this was created by examining a subset of inputs and outputs in ignorance of whether they are passing or failing pairs, and then applied automatically in the reminder of the data set. The coding scheme for execution traces was the same algorithm used by [8] and also has no information about whether a trace is associated with a passing or failing execution.

VII. Conclusion and Future Work

In this paper we present an empirical evaluation study, which investigates the effectiveness of automated test oracles based on unsupervised/semi-supervised machine learning techniques and dynamic analysis techniques (i.e. Daikon). We perform three studies; each study examines possible practical scenarios to construct automated test oracles based on machine learning techniques with two different sets of dynamic execution data (input/output pairs only or input/output pairs augmented with execution traces) as follows: (1) For unsupervised learning scenario, testers have test data but they do not have advance knowledge about passing and failing outcomes; (2) For semi-supervised learning scenario, testers have test data and they are able to label both normal (passing) and abnormal (failing) tests, or label normal (passing) tests alone; (3) Testers use the cheap results of unsupervised learning techniques as input to semi-supervised/supervised learning techniques in the case of no label tests available.

Results for the first empirical investigation (input/output pairs only used as input to algorithms) demonstrate that Daikon as test oracle has the highest fault detection rate compare to test oracles based on unsupervised and semi-supervised learning techniques (the fault detection rate is approximately 64% for Daikon, 61% for unsupervised learning techniques and 43% for semi-supervised learning techniques on average for all systems). However, test oracles based on machine learning techniques have slightly lower false positive rate compare to Daikon (the false positive rate for machine learning techniques is 3% lower on average for all systems). In addition, the results for the second empirical investigation (input/output pairs and execution traces used together as input to algorithms) show that automated test oracles based on semi-supervised learning techniques have the lowest false positive rate compare to unsupervised learning techniques and Daikon (approximately 11% on average for all systems), and also the highest fault detection ability (approximately 82% on average for all systems). Furthermore, automated test oracles based on unsupervised learning techniques have slightly higher fault detection ability compared to Daikon (the fault detection rate is roughly 66% for unsupervised learning techniques and 64% for Daikon on average for all systems), but a similar false positive rate (approximately 16% on average for all systems). However, there is some considerable variability between systems and a significant part of our future work in this are will focus on reducing this to make the approach practically viable.

Despite of the initial achievements of test oracles based on machine learning techniques, there are number of barriers for the approaches to become practically usable which fall into the categories of scalability and accuracy. Both barriers is explained further below:

A. Improving Accuracy

Accuracy in this context means the ability of the presented approaches to identify failing and passing test results as correctly as possible (high true positive rate and low false positive/false negative rates). The accuracy of semi-supervised and unsupervised learning techniques may be improved by augmenting the data sets (input/output pairs and execution traces) with more relevant information from the program execution (e.g. state information, execution time and code coverage etc.) to build more effective/accurate test oracles.

Adding more execution data to the data sets can help to reduce the size of labelled data used to train the learning algorithms in semi-supervised learning. This also relates to scalability but to make the approach practical the size of labelled data needs to be as low as possible which means improving the accuracy as well. The other point related to the size of labelled and then accuracy is that the predictions of semi-supervised learning approaches should come with an estimate of confidence, possibly associated with the proportion of labelled data used.

The accuracy of unsupervised learning approaches can be improved by selecting the most appropriate similarity measures for clustering algorithms. In addition, the number of specified clusters for clustering algorithms is important and the accuracy can be improved by specifying the optimal number of cluster counts. Note that, the accuracy of unsupervised learning techniques (clustering algorithms) in this paper means the separation between failing and passing test results. In other words, the failing test results should be grouped in small clusters with high failure density compared to large clusters which should have more passing test results. The definition of 'small' and 'large' is quite coarse in this context. Finding the appropriate definition of 'small' and 'large' clusters can help to improve accuracy in practice, as well as providing some guidelines to the tester on the point where is not worth exploring further clusters.

B. Increasing Scalability

The test data transformation for the software under test is a very important issue which affects the scalability of the presented approaches in this paper. This clearly impacts on the volume of data that has to be processed but also has implications for accuracy too, so must be done in a way that does not compromise this. To be practically applicable it is necessary to find a generic automated approach for each type of test data. For instance, the input/output pairs for tested systems in this paper were string/text type and it turned out that tokenization procedure worked reasonably well with the presented approaches but this may not be generally applicable for all input and output types.

Generally, further research is needed to overcome those barriers by conducting further empirical investigation of the effectiveness of presented approaches to corroborate the findings and to increase their external validity, particularly by exploring a wider range of programs, faults and coding schemes for dynamic execution data (input/output pairs and execution traces etc.). In summary, the fundamental principle to the successful automated test oracles is the capability to build oracles that demonstrate a substantially lower false positive rate and higher fault detection capability, as compared to the available, state of the art tools. Our future research will be also devoted to further development and empirical investigation of the effectiveness of several automated test oracles, to evaluate the features offered by different alternative oracles.

References

[1] R. Almaghairbe and M. Roper, "Automatically classifying test results by semi-supervised learning," in *27th IEEE International Symposium on Software Reliability Engineering, ISSRE 2016, Ottawa, ON, Canada, October 23-27, 2016*, 2016, pp. 116–126. [Online]. Available: https://doi.org/10.1109/ISSRE.2016.22

[2] ——, "Separating passing and failing test executions by clustering anomalies," *Software Quality Journal*, vol. 25, no. 3, pp. 803–840, 2017. [Online]. Available: https://doi.org/10.1007/s11219-016-9339-1

[3] M. Roper, "Using machine learning to classify test outcomes," in *IEEE International Conference On Artificial Intelligence Testing, AITest 2019, Newark, CA, USA, April 4-9, 2019*, 2019, pp. 99–100. [Online]. Available: https://doi.org/10.1109/AITest.2019.00009

[4] M. D. Ernst, J. H. Perkins, P. J. Guo, S. McCamant, C. Pacheco, M. S. Tschantz, and C. Xiao, "The daikon system for dynamic detection of likely invariants," *Sci. Comput. Program.*, vol. 69, no. 1-3, pp. 35–45, Dec. 2007. [Online]. Available: http://dx.doi.org/10.1016/j.scico.2007.01.015

[5] G. J. Myers, C. Sandler, and T. Badgett, *The Art of Software Testing*, 3rd ed. Wiley Publishing, 2011.

[6] R. Oliveira, U. Kanewala, and P. Nardi, *Automated Test Oracles: State of the Art, Taxonomies and Trends*, 10 2014, vol. 95, pp. 113–199.

[7] G. Fraser and A. Arcuri, "Evosuite: Automatic test suite generation for object-oriented software," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ser. ESEC/FSE '11. New York, NY, USA: ACM, 2011, pp. 416–419. [Online]. Available: http://doi.acm.org/10.1145/2025113.2025179

[8] C. D. Nguyen, A. Marchetto, and P. Tonella, "Automated oracles: An empirical study on cost and effectiveness," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2013. New York, NY, USA: ACM, 2013, pp. 136–146. [Online]. Available: http://doi.acm.org/10.1145/2491411.2491434

[9] E. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *Software Engineering, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2014.

[10] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 15:1–15:58, Jul. 2009. [Online]. Available: http://doi.acm.org/10.1145/1541880.1541882