Cluster-Based Parallel Testing Using Semantic Analysis

Cristina Landin[†], Sahar Tahvili[*‡], Hugo Haggren[*], Martin Längkvist[†], Auwn Muhammad[*], Amy Loutfi[†]
[*]Global Artificial Intelligence Accelerator (GAIA), Ericsson AB, Stockholm, Sweden
{sahar.tahvili, hugo.haggren, auwn.muhammad}@ericsson.com
[†]School of Science and Technology, Örebro University, Örebro, Sweden
{cristina.landin, martin.langkvist, amy.loutfi}@oru.se
[‡]School of Innovation, Design and Engineering, Mälardalen University, Västerås, Sweden

***Abstract*—Finding a balance between testing goals and testing resources can be considered as a most challenging issue, therefore test optimization plays a vital role in the area of software testing. Several parameters such as the objectives of the tests, test cases similarities and dependencies between test cases need to be considered, before attempting any optimization approach. However, analyzing corresponding testing artifacts (e.g. requirement specification, test cases) for capturing the mentioned parameters is a complicated task especially in a manual testing procedure, where the test cases are documented as a natural text written by a human. Thus, utilizing artificial intelligence techniques in the process of analyzing complex and sometimes ambiguous test data, is considered to be working in different industries. Test scheduling is one of the most popular and practical ways to optimize the testing process. Having a group of test cases which are required the same system setup, installation or testing the same functionality can lead to a more efficient testing process. In this paper, we propose, apply and evaluate a natural language processing-based approach that derives test cases' similarities directly from their test specification. The proposed approach utilizes the Levenshtein distance and converts each test case into a string. Test cases are then grouped into several clusters based on their similarities. Finally, a set of cluster-based parallel test scheduling strategies are proposed for execution. The feasibility of the proposed approach is studied by an empirical evaluation that has been performed on a Telecom use-case at Ericsson in Sweden and indicates promising results.**

***Index Terms*—Software Testing, Natural Language Processing, Test Optimization, Semantic Similarity, Clustering**

I. Introduction

Software testing can be considered as the most critical, labor-intensive and time-consuming process in the software development life cycle (SDLC), which almost demands 50% of the total development cost [1]. Therefore, test optimization techniques e.g. test case selection, prioritization, scheduling has received a great deal of attention in recent years. Executing all designed test cases inside of the test suite without a specific plan can lead directly to the waste of testing resources [2]. On the other hand, efficiently scheduling test cases for execution is also a time-consuming process where several factors such as execution time, requirement coverage, dependency and similarities between test cases need to be analyzed in an early stage of a testing process [3], [4]. There is a trade-off between the required effort for scheduling test cases and executing them without any specific order. Having an accurate and fast approach for measuring the mentioned testing factors and also scheduling test cases for execution based on them can help test developers and software engineers in a test team to save time and also testing resources. Moreover, understanding the test case properties (mentioned factors) can be utilized for other test optimization purposes such as test suite reduction, parallel test execution and also test automation. However, both test cases properties and test optimization techniques can take substantial time and effort in a manual testing procedure, where all testing artifacts (e.g. requirements, test cases) are written by a human in a natural text. Additionally, the complexity of the product under test, the company's infrastructure should be considered for selecting a feasible test optimization approach. For instance, parallel test execution is a suitable approach where multiple products, applications or sub-components of one application need to be tested in one testing environment. However, in other cases, serial testing is a more practical test execution approach. In this paper, we proposed a cluster-based parallel test execution for manual integration test cases. The proposed approach is an NLP-based approach that detects the similarities between manual test cases. Later, we spot test cases that are structurally identical and can be considered as duplicate and also those test cases which are very similar or partially similar to each other. Moreover, we calculate the similarity between test cases based on their test specifications, using the Levenshtein distance. The proposed approach in this paper opens ways to execute test cases in a more efficient way, where the semantically similar test cases between different products are going to be tested parallel at the same time. Finally, the feasibility of the proposed approach is analyzed on an industrial case study in the Telecom domain at Ericsson in Sweden. This paper makes the following contributions:

1) Measuring the similarities between manual integration test cases.
2) Eliminating possible duplication of test cases.
3) Proposing a cluster-based parallel approach for test execution.

II. Background

Parallel test execution can be considered as a potential solution for reducing testing time by increasing the utilization of

This work has been supported by the Swedish Knowledge Foundation (KKS) and VINNOVA through CoAIRob industrial research school and TESTOMAT project respectively.

978-1-7281-6984-2/20/$31.00 ©2020 IEEE
DOI 10.1109/AITEST49225.2020.00022

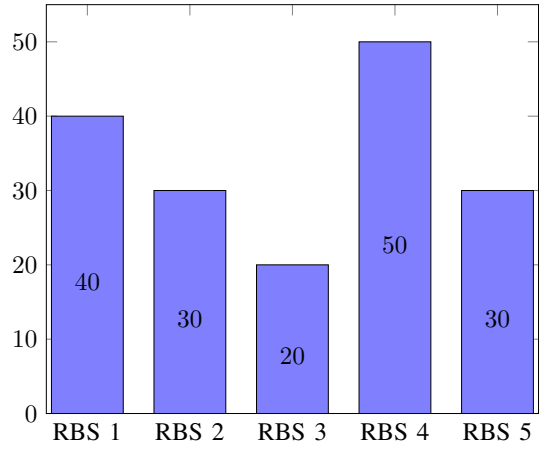

Figure 1: An overview of the serial execution process for testing five RBSs at Ericsson. The X-axis shows the name of the products and the Y-axis represents the execution time.

available resources (e.g. CPU, processor cores, instrumentation resources, microprocessors) [5] and it might result in faster time to delivery of the final product. Testing the similar functionality between several products in parallel across physical and virtual machines can resolve the limitations of time and budget while still assuring quality [6]. On the other hand, parallel testing has its own sets of challenges and it is usually applicable in large industries where there are no infrastructure or test station limitations. The software quality assurance process at Ericsson is a costly process where all radio products need to be tested for compliance with relevant regulations and standards on electromagnetic fields before they are delivered to the market [7]. Radio Base Station (RBS) is an important key element on mobile communication [8] among Ericsson's products. Their inherent complex design makes their testing process very advanced to follow all the standards, especially when an old and new standard needs to be covered at the same time. The standards are translated as test cases which are included several test steps. To a large extension, manual work and subject matter expert (SME) knowledge are still required to translate the standards to test cases to test their requirements. Figure (1) shows a general overview of testing 5 different RBSs at Ericsson. As we can see, all RBSs are tested in series and the total required time for testing them varies from product to product. The number of generated test cases for testing each RBS and other factors such as RBS's properties and design can impact the required time for testing an RBS. Moreover, each RBS will be tested in a specific test station. A typical test station at Ericsson is the hardware interface between the RBS and the test engineer. Today test cases are mostly executed sequentially, which can be considered as the most expensive way of test execution [9], which cannot even guarantee the best execution time for the test suite. The approximate time for a serial testing procedure can be defined as [9]:

$$t \approx \sum_{i=1}^{n} t_i \tag{1}$$

where t_i is the execution time of each ith test case and n represents the total number of test cases. Additionally, t_i is a sum of several factors such as system setup time, installation time and the required time for executing a test case. As highlighted earlier, executing similar test cases in parallel is a way to reduce the total testing time, thus [9]:

$$t = \frac{\sum_{i=1}^{s} t_i}{p} + \sum_{i=s+1}^{n} t_i \tag{2}$$

where s represents similar test cases and p is the parallel test stations for executing test cases in parallel. We need to consider that, similar test cases are distributed across different RBSs. While non-similar test cases are executed successfully, then similar test cases can be executed in parallel at any given time. Furthermore, identifying similar test cases can lead to a more efficient usage of testing resources through:

- To figure out which test cases are superfluous and can be eliminated.
- Simultaneous execution of test cases that test the same functionality or required the same precondition, system setup, and installation.
- Any combination of the previous options.

The preconditions for a test case include the state that a system and its environment must be reached before a specific test activity can be run. In other words, the preconditions specify the setup that is needed for test execution. Many possible types of settings and conditions can be interpreted as a precondition but, generally, a precondition can fall under one of the following classes:

- A previously executed step in the testing sequence.
- The outcome of the previous step in the testing sequence.
- The availability of existing data needed to run the test case.

Although parallel test execution can significantly reduce the testing time, however, not special efforts have been put into proposing a suitable approach yet. It could be due to many reasons e.g. test cases dependencies and infrastructure limitations.

III. Related work

Parallel testing for hardware and software applications has been studied during years. Karimi and Lombardi [10] present an approach to reduce the test time complexity based on the functional parallelism of a multi-port Random Access Memory with many ports. In that way reducing the number of tests based on the common patterns found in the design of the component. Delgado [11] propose a new technique to reduce the test time in test program execution, through parallel testing in the three following scenarios: multiple units of the same product, multiple units of different products and reducing test coverage using: test pipe-lining architecture, scheduling method and reducing the number of tests without jeopardizing the quality of the product, respectively. Both approaches mentioned above are based on specification-based testing [12]. In other words, they do not try to reduce the testing time by modifying the specification requirements, instead, they follow the constraints. Misailovic et al. [13] use parallel test

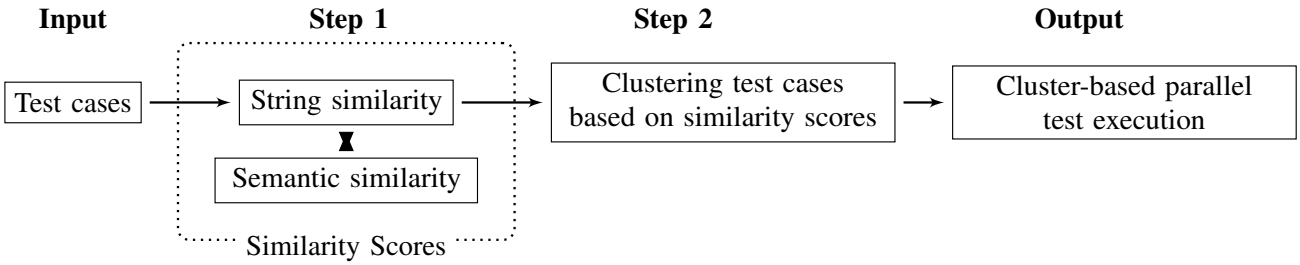

Figure 2: The steps of the proposed solution for cluster-based parallel execution using semantic similarity between test cases.

generation and execution to generate and validate the test cases given by the constraints for structurally complex test inputs. The tool automatically generates fewer equivalent inputs given the constraints (bounds). Correia et al. [14] demonstrate how their tool called MOTSD works in the industrial context of OutSystems. MOTSD uses a Particle Swarm Optimization (PSO) algorithm and the goal is to minimize the regression testing and diagnostic costs by providing faster feedback to the developers when a change is made. MOTSD runs a subset of the test suite close related to the change, reduces the costs of fault location. Gonzalez-Sanchez et al. [15] present a spectrum-based fault localization-approach for test prioritization. in this approach, those test cases which are more likely to fail using a heuristic indicator will be executed first. However, the proposed approach in [15] is applicable in a more mature state of the findings, that the test suite can be reduced by testing only the most important features in order to reduce the cost of failure localization. In the same way, Vahabzadeh et al. [16] utilize an open-source tool called TESTLER to tackle the test suite minimization based on removing the partial redundant test statements. The issue with this approach is that is only applicable on test code analysis and not test cases described in text. We aim to generate fewer equivalent inputs to produce a suitable test suite without compromising the quality of the test based on the semantic similarity of the requirements for a multiple port unit, multiple units, and different products. Approximate string matching [17] is the task of comparing two strings and approximate the similarity between them. In the literature, it is used for a wide range of applications, including spell checking and optical character recognition correction [18], pattern recognition [19], spam filtering [20], and record linkage [21]. The use of measurement for comparing two strings can differ depending on the application and some proposed error models are the episode distance, q-gram distance, longest common subsequence (LCS), Hamming distance, block distance, etc. The Levenshtein distance (or edit distance) between two strings is a commonly used measurement for approximate string matching and is the minimum amount of character alterations (insertions, deletions, and substitutions where each alteration cost 1) that are needed to make the two strings the same. The lower bound of the Levenshtein distance $d(x, y)$ for two strings x and y is 0 and the upper bound $\max(|x|, |y|)$. It has previously been used in the linguistic distance to measure the difference between two languages [22]. For longer strings, like sentences or documents, other textual analysis techniques have been proposed in the literature. Latent semantic analysis (LSA) performs a quantitative content analysis on a set of documents by counting the prevalence of each word in each document and then group similar documents that share the same words [23]. It has been used for search engine procedures and information retrieval applications [24], textual analysis [25], gene expression analysis [26], central bank communications [27] and corporate social responsibility (CSR) analysis [28]. More recently, word embeddings have been learned by using neural network architectures for measuring semantically similar words. The similarity between the two words is then computed by calculating the cosine similarity between their word embeddings. Two popular methods for word embeddings are word2vec [29] and GloVe [30]. For measuring the similarity between sentences using word embeddings there are a couple of techniques proposed, such as calculating the cosine similarity between the average of the word embeddings of all words in the two sentences, Word mover's distance [31], and Smooth Inverse Frequency [32] that weights the word embeddings by their corpus prevalence and removes irrelevant words. However, the above methods do not take word order into account. Some generic sentence encoders that preserve word order are doc2vec [33], Skip-thought vectors [34], InferSent [35], BERT [36], and Universal sentence encoder [37]. In this paper, we use the Levenshtein distance to compare the test cases. This is due to a large number of test cases that need to be compared and the structure of the test cases were word order is not important. To the best of our knowledge, this work is the first that explores the use of test description analysis to perform product test optimization.

IV. Method

In this paper, we propose, apply and evaluate an NLP-based approach for execution similar test cases in parallel. Figure (2) represents an overall overview of our proposed approach. According to Figure (2) the required input for utilizing the proposed approach is the test case specifications, which is written in the natural text by the testers. In the next step, the semantic similarities between test cases will be measured by using LSA techniques, edit distance metrics. Later, test cases will be categorized into several groups based on their semantic similarities. Finally, the test cases which are similar to each other need to be executed in parallel on the same test stations. In this paper, we define the concept of semantic similar test cases as:

Definition 1. *Test case TC_1 and TC_2 are semantically similar if they are designed to test the same functionality or they have the same preconditions, execution requirements (installation), system setup.*

A. Semantic Analysis Using Edit Distance Metrics

Edit distance algorithms are a common approach to recognize textual similarities. The basic operations used to edit are insertion, substitution, and deletion. Each operation has its associated cost, this means that some operations are more expensive than others. We minimize this cost via ML algorithms and heuristics. The cost of a pair of texts based on how many operations were necessary to transform one text into another text is used as an edit distance score. Using edit distance metrics techniques for analyzing and parsing test cases, can be considered as an appropriate approach for measuring the semantic similarities between test cases. In this paper, we use the Levenshtein distance which is a string metric for computing the difference between two sequences, proposed by Vladimir Levenshtein [38].

Definition 2. *The Levenshtein distance between two strings is the minimum amount of character alterations that need to be made to turn the one string into the other.*

By alterations, we refer to substitutions, additions, and removals. Mathematically, the Levenshtein distance algorithm can be described as [39]:

$$l_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} l_{a,b}(i-1,j)+1 \\ l_{a,b}(i,j-1)+1 \\ l_{a,b}(i-1,j-1)+\mathbf{1}_{a_i \neq b_j} \end{cases} & \text{otherwise.} \end{cases} \tag{3}$$

where a and b are two strings to be compared and i and j are the current index we are evaluating, starting at $|a|$ and $|b|$ respectively. Furthermore, the theoretical maximum of the Levenshtein distance is the length of the longer string, since if the strings have no characters in common you have to first substitute all the characters in the shorter string, then add the rest. This upper limit allows us to define a ratio of similarity between the two strings a and b as following [39]:

$$l_{ratio}(a,b) = 1 - \frac{l_{a,b}(|a|,|b|)}{\max(|a|,|b|).} \tag{4}$$

As mirrored in Figure (2), applying the similarity detection techniques (e.g. Levenshtein distance measurement) on the test cases provides string similarity scores as an output, which can be utilized for both clustering and classification purposes.

B. Cluster-Based Test Scheduling Strategies

In this subsection, we propose a set of cluster-based test case scheduling strategies based on the semantic similarities between test cases. By close consultation with SMEs at Ericsson, we decide to propose the following clusters "Identical", "Very Similar", "Similar" and "Partially Similar", for grouping test cases based on their semantic similarities. Table (II) gives an example for each mentioned cluster, using manual integration test cases generated by testers at Ericsson. In this example, the semantic relationship of TC_1 is analyzed with TC_2. As we can see, there are four different versions of TC_2. At first sight, all different versions of TC_2 look similar, therefore having an NLP-based approach can help us to distinguish all different versions and also measuring the semantic relationship of it with other test cases. In the presented example in Table (II), both TC_1 and TC_2 are designed to test the same functionality and both require the same system set up, installation and preconditions. Moreover, we inserted a threshold in Table (II) is our assumption for dividing test cases into each cluster. In other words, if the Levenshtein distance between two test cases is equal to 1, thus the test cases are identical. The proposed threshold for other clusters is our assumption, which has been made in this paper. However, the proposed approach in this paper is not limited to the mention clusters and thresholds, where any other label, name or threshold can be utilized. As stated before, those test cases which are in the same cluster should be executed parallel, since, the semantic similar test cases are usually required the same system setup, installation or test the same functionality.

V. Industrial Case Study

The provided industrial case study in this work is following the proposed guidelines for conducting and reporting case study research in software engineering by Runeson and Höst [40] and specifically the way guidelines are followed in [41] and [2].

Property	RBS 1	RBS 2	RBS 3	RBS 4	RBS 5
LTE	✓	✓	✓	✓	✓
WCDMA	✓	✓		✓	✓
GSM	✓	✓		✓	✓
TDD	✓			✓	
FDD	✓	✓	✓	✓	✓
Ports	4	2	2	4	2
Number of test cases	87	78	77	91	84

Table I: Selected RBS, number of test cases and relevant properties.

In this study, five multi-standard RBSs compatible with at least 3G/WCDMA, 4G/LTE and 5G are selected as a case under study. These products are intended to cover different markets and operate with different standards, e.g. 2G/Global System for Mobile Communication (GSM), 3G/Wideband Code Division Multiple Access (WCDMA) and 4G/Long-Term Evolution (LTE), both for Time Division Duplex (TDD) and Frequency Division Duplex (FDD). Table (I) shows the number of assigned test cases and also the designed properties, technologies for each RBS. According to Table (I) the number of designed test cases for each RBS is different, which may indicate directly to the correlation to the required time for testing each RBS, mirrored earlier in Figure (1).

A. Unit of Analysis and Procedure

The units of analysis in the case under study are test cases, extracted from an internal database at Ericsson. Each test case is written in a natural text by SMEs at Ericsson and contains e.g. tens of hundreds of test steps inside of each test case. The case study is performed in several steps:

1) A total number of five products (RBS) are selected as a case under study.

Cluster	Test case example	Threshold
Identical	TC_1: This test case will measure the current status of a LED. TC_2: This test case will measure the current status of a LED.	Levenshtein distance $= 1$
Very Similar	TC_1: This test case will measure the current status of a red LED. TC_2: This test case will measure the current status of a green LED.	$0.95 \leq$ Levenshtein distance ≤ 0.99
Similar	TC_1: This test case will measure the current status of a LED. TC_2: This test case will measure the current status of a diode.	$0.91 \leq$ Levenshtein distance ≤ 0.94
Partially Similar	TC_1: This test case will measure the current status of a LED. TC_2: This test case will bias a LED and measure the current.	$0.8 \leq$ Levenshtein distance ≤ 0.9

Table II: Some examples of identical, very similar and partially-similar test cases. LED stands for Light Emitting Diode.

2) A total number of 417 test cases are captured from the Ericsson's database for each product (RBS).
3) The Levenshtein distance is measured between all test cases, where $(417 \times 416) \div 2 = 86736$ comparisons are made between test cases. In other words, the Levenshtein distance is measured between every single test case with all other test cases between five RBSs.
4) Test cases are divided into four different clusters based on their semantic similarities.
5) A set of cluster-based test case scheduling strategies based on the semantic similarity between test cases are proposed by us.

Figure (3) represents the obtained results using test cases for five different products at Ericsson, clustered into four groups based on the semantic similarities (the Levenshtein distance) between test cases.

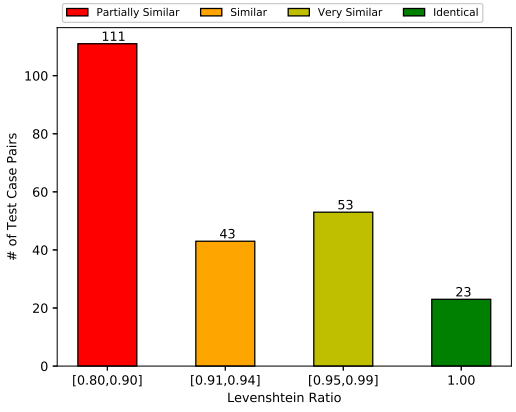

Figure 3: The clustered test case pairs into four groups using Levenshtein distance.

As can be seen, all test cases (presented in Table (I)) are divided into four clusters based on their semantic similarities relationship. The number of test cases presented in Figure (3) represents test case pairs. For instance, 111 test case pairs are classified into the "Partially Similar" cluster. Thereby, in total, the number of 246 test cases are grouped into mentioned similar clusters, where the total number of 171 test cases have remained as non-semantic similar test cases within all five products (RBS).

B. Model Performance Evaluation

Evaluation metrics explain the performance of a model. Selecting a suitable performance metrics is critical and also influences the measured performance of the model [42]. To evaluate the performance of the proposed approach in this paper, a Ground Truth (GT) is conducted using the SME's knowledge at Ericsson, by manually labeling 20% of the utilized test cases in this study.

C. Evaluation of Binary Classifiers

Utilizing of binary classifiers for evaluation can help us to learn a model which tries to predict whether some instance belongs to a class or not [43]. Moreover, the output of many clarification algorithms such as logistic regression and decision trees is a probability for an instance belonging to a positive class.

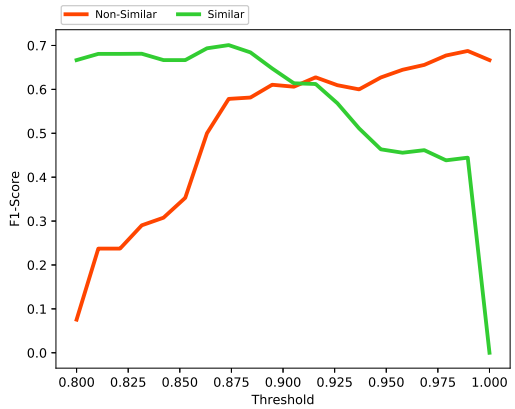

Figure 4: F1-score for similar and non-similar test cases and the threshold for detecting the optimal point.

In this paper, we measure F1-score as a proper evaluation metric to show the performance of the proposed solution. F1-score represents a harmonic relationship between precision and recall. In this case, the precision and recall can be defined as follow:

- **Precision:** is the number of correctly detected similarities over the total number of detected similarities by us.

- **Recall:** is the number of correctly detected similarities over the total number of existing similarities.

Thereby:

$$F1 - Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (5)$$

However, both precision, recall, and also F1-Score need to be measured for non-similar test cases as well, using Equation (5). Figure (4) shows similar and non-similar test cases and according to F1-scores of the clustering task whether test case TC_1 matches the according to the GT or not. As can be observed in Figure (4), the F1-score for both similar and non-similar class behaves as one would expect. The F1-score for the non-similar class increases as we raise the threshold, while it decreases for the similar class. The optimal threshold can be observed to lie where the F1-score is the highest for both classes, namely at 0.91 with an F1-score of 0.61. However, we need to consider that, the utilized GT for measuring F1-score is provided by the SMEs, which might suffer from wrong labeling, uncertainty, and ambiguity, because of human judgment.

VI. Result

As presented in Figure (3), all test cases are divided into 4 clusters based on their similarities, where the remaining test cases are non-similar. Figure (5) shows the number of clustered test cases per product (RBS), where similar test cases between 5 products can be executed at the same time.

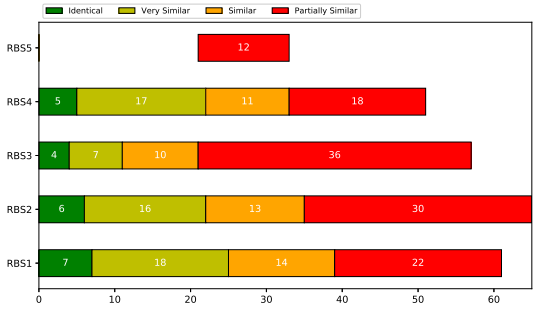

Figure 5: An overview of the parallel execution process for testing five RBSs at Ericsson. The x-axis represents a calendar time slot, the numbers inside the bars are the number of test cases per cluster.

According to Figure (5), the number of test cases per cluster is different, where e.g. RBS5 has just 12 test cases which are "Partially Similar" to other test cases in other RBSs. It means that 72 test cases which are designed to test RBS5 (see Table (I)) can be executed serially, before or after parallel execution of those 12 partially similar test cases. However, other RBSs have also other non-similar test cases, which need to be executed serially, but as we can see in Figure (5), most of the test cases can be executed in parallel. As stated in Section (II), parallel test execution can help the testing team for saving more time and testing resources. For instance, executing all (in total 22) identical test cases (see Figure (5)) between five RBSs at one testing station is an efficient decision. The testers need to set up the testing environment for testing once and several test cases can be executed. However, in some testing situations, not all identical test cases need to be executed, where getting a "Pass" answer for some of them might be sufficient. However, the following assumptions need to be made or satisfied for a parallel test execution:

1) There are no dependencies between test cases.
2) All test cases have the same execution time (t_{aj}).
3) The required time for system setup, installation and getting the testing station ready for testing is the same (t_{cj}).
4) There are at least $np + 1$ available test stations, to make parallel testing feasible, where np is the number of products tested.

Additionally, the total execution time (T_{total}) for testing a test case can be calculated as:

$$T_{total} = \sum_{j=1}^{n} t_{cj} + t_{aj} \quad (6)$$

Where t_{cj} is the system setup time and t_{aj} indicates to the test cases execution (running) time [44]. Parallel test execution for those test cases which have the same system setup. Thus, the saving time (ST) for those test cases which are required the same system setup can be calculated as:

$$ST = \frac{(np - 1)}{np} \times 100 \quad (7)$$

Using the presented results in Figure (5), Equation (7) and making the mentioned assumptions, we can estimate the total saving time using parallel test execution for the five products. In this regard, we propose four different execution scenarios which are matched the similarities clusters.

Table III: Time-saving potential for different parallel execution scenarios for five RBSs. All test cases have the same t_{cj}.

Scenario	Cluster	Time saving
1	Identical test case	95.4%
2	Very similar test cases	98.2%
3	Similar test cases	97.9%
4	Partially similar test cases	99.1%

Table (III) represents the potential time saving for five RBSs using different parallel execution scenarios. To apply the proposed scenarios, in our case, we need to have at least four available test base stations. In other words, each cluster of test cases (distributed between five RBSs) should be executed in one testing base station, thus the system setup, installation, and pre-conditions will be installed one time for executing several test cases. In the presented scenarios in Table (III), we roughly estimated the total saving time, when the system is seated once at each testing station and all test cases are executed one after another. For instance, in total 22 test cases are detected as identical test cases (see Figure (5)). According to Scenario 1, all those 22 test cases should be executed at one test station. Thus, if all test cases are required the same system setup and installation effort ($t_{cj_1} = t_{cj_2} = ... = t_{cj_{22}}$), it will be sufficient

to perform the system setup activities just for one test case and execute the remaining 21 test cases. Furthermore, minimizing t_{cj} can lead to minimizing the total test execution time (t_{total}), presented in Equation (6). We need just to consider that, the proposed approach and analyzed scenarios in this paper, are just reduced t_{cj}. Moreover, testing the same functionality is also required the same pre-condition and installation, which can be saved by running similar test cases in parallel. To avoid any confusion, we just clustered those test cases which have the Levenshtein distance greater than or equal to 0.8, which is a high value. However, as presented in Table (II), those test cases which are clustered as similar, are designed to test the same function and also required the same system setup, installation effort.

VII. Threats to validity

The major construct validity threat [45] in this study is the way that the semantic similarities between test cases are measured. Utilizing just the test specifications for similarity detection may not be sufficient in other testing processes. Sometimes analyzing other testing artifacts such as requirement specifications, standers and test records might help us to detect the semantic similarities as well. Moreover, the proposed threshold in this study is selected by close consultation with SMEs, which need to be measured more precisely. The main limitation of the proposed approach in this paper is the infrastructure and testing station. For applying the proposed approach in this paper, we need to have several available test stations for testing each product parallel with others. The coming technologies as 5G will only enlarge those challenges due to the increase in the number of elements i.e. the number of ports, making this solution a necessity.

VIII. Discussion and Future Work

The main goal of this study is to propose and apply an approach that first detects the semantic similarities between manual integration and later cluster test cases for execution. To this end, we make the following contributions:

- We have proposed an NLP-based approach to detect the semantic similarities between manual integration test cases automatically. The semantic similarities have been extracted by analyzing test case specifications.
- Test cases have been grouped into several clusters (e.g. Identical, Very similar, Similar and Partially similar) based on their semantic similarities.
- A set of cluster-based parallel test execution was proposed.
- The feasibility of the proposed approach in this paper was analyzed on an industrial use case at Ericsson.

Parallel test execution can provide an opportunity for using the testing resources more efficiently. Decreasing the redundant efforts for system setup, installation and preparing the testing environment ready for testing, directly impacts the testing total cost and quality. Executing test cases in a parallel way might help testers and test managers to save more time and cost for testing a set of large products. On the other hand, having an applicable, effective and efficient solution for parallel test execution is a challenging task, where the trade-off between the required effort and possible gain should be analyzed in an early stage of a testing process. The product's complexity, properties, quality, and testing level have a direct effect on the number of required test cases for testing a product. However, usually, a large set of test cases are designed to testing a product at a high-level testing level such as integration testing. Since the required input for applying the proposed approach in this paper is just test specification, thus the approach can be considered as an applicable approach in all industries, which work with manual testing.

A. Future Work

Developing the proposed approach as a tool that can handle even larger sets of test specifications is one of the future directions of the present work. Previously [46], [47] we applied and evaluated another LSA method on a large industrial case study. Comparing the performance of the proposed solution in this paper with other LSA methods is also another considered research direction for us. In the future, one more step will be added to the proposed approach (see Figure (2)), which will check the requirement specification behind each test specification. The main goal by adding this step is to detect the dependencies between test cases (if any exists). In this paper, we assumed that there are no dependencies between test cases and the order of execution inside of each product is not important, while these assumptions might not be valid for all applications [48], [49]. Hitherto, we proposed several approaches for detecting the dependencies between manual integration test cases, where several testing artifacts such as signal information [50] and requirement specification [51] have been used. In future applications, a method for threshold determination could be of interest. One possibility would be to use evaluation as a tool by choosing the threshold with the best evaluation metrics, i.e. the threshold that best aligns with SME-judgment.

IX. Conclusions

Test optimization can be considered as a key role player in the software development life cycle and it can be performed through a different way of test scheduling, test case selection, prioritization and also automation. In this paper, we introduced and applied our proposed NLP-based approach, for scheduling manual integration test cases for execution in a parallel way. Our proposed approach takes test specifications as input and provides a cluster-based parallel test execution as output. First, the Levenshtein distance between test cases is measured and secondly, test cases are grouped for execution based on their semantic similarities. Our empirical evaluations at Ericsson AB and analysis of the results of one industrial project show that the proposed approach is an applicable solution for scheduling test cases for execution in a parallel way. The proposed approach in this paper is also able to handle a large set of manual test specifications, where the proposed clustering can be used for other purposes such as test automation.

REFERENCES

[1] M. Pezzè and M. Young, *Software Testing and Analysis: Process, Principles, and Techniques*. Wiley, 2008.

[2] S. Tahvili, "Multi-criteria optimization of system integration testing," Ph.D. dissertation, Mälardalen University, December 2018.

[3] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler, "Scalable multi-objective optimization test problems," in *Proceedings of the 2002 Congress on Evolutionary Computation.*, 2002.

[4] S. Tahvili, M. Bohlin, M. Saadatmand, S. Larsson, W. Afzal, and D. Sundmark, "Cost-benefit analysis of using dependency knowledge at integration testing," in *The 17th Int. Conf. On Product-Focused Software Process Improvement*, 2016.

[5] S. Delgado, "Parallel testing techniques for optimizing test program execution and reducing test time," in *IEEE AUTOTESTCON*, 2008.

[6] Y. Liu, T. Jiang, and P. Lin, *Optimal Testing Resources Allocation for Improving Reliability Assessment of Non-repairable Multi-state Systems*. Springer International Publishing, 2018, pp. 241–264.

[7] A. Najafi, W. Shang, and P. Rigby, "Improving test effectiveness using test executions history: An industrial experience report," in *41st Int. Conf. on Software Engineering: Software Engineering in Practice*, 2019.

[8] E. Hossain, V. Bhargava, and G. Fettweis, *Green Radio Communication Networks*. Cambridge University Press, 2012.

[9] A. Lastovetsky, "Parallel testing of distributed software," *Information and Software Technology*, vol. 47, no. 10, pp. 657–662, 2005.

[10] F. Karimi and F. Lombardi, "Parallel testing of multi-port static random access memories for bist," in *International Symposium on Defect and Fault Tolerance in VLSI Systems*, 2001.

[11] S. Delgado, "Parallel testing techniques for optimizing test program execution and reducing test time," 2008.

[12] P. Stocks and D. Carrington, "A framework for specification-based testing," *IEEE Transactions on Soft. Engin.*, vol. 22, no. 11, pp. 777–793, 1996.

[13] S. Misailovic, A. Milicevic, N. Petrovic, S. Khurshid, and D. Marinov, "Parallel test generation and execution with korat," in *ESEC*, 2007.

[14] D. Correia, R. Abreu, P. Santos, and J. a. Nadkarni, "Motsd: A multi-objective test selection tool using test suite diagnosability," in *Proceedings of the 27th Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019.

[15] A. Gonzalez, É. Piel, R. Abreu, H. Gross, and A. van Gemund, *Prioritizing Tests for Fault Localization*. Springer, 2013, pp. 247–257.

[16] A. Vahabzadeh, A. Stocco, and A. Mesbah, "Fine-grained test minimization," in *40th Int. Conf. on Software Engineering (ICSE)*, 2018.

[17] G. Navarro, "A guided tour to approximate string matching," *ACM computing surveys (CSUR)*, vol. 33, no. 1, pp. 31–88, 2001.

[18] T. Lasko and S. Hauser, "Approximate string matching algorithms for limited-vocabulary ocr output correction," in *Document Recognition and Retrieval VIII*, 2000.

[19] A. Marzal and E. Vidal, "Computation of normalized edit distance and applications," *IEEE transactions on pattern analysis and machine intelligence*, vol. 15, no. 9, pp. 926–932, 1993.

[20] F. Zhou, L. Zhuang, B. Zhao, L. Huang, A. Joseph, and J. Kubiatowicz, "Approximate object location and spam filtering on peer-to-peer systems," in *International Conference on Distributed Systems Platforms and Open Distributed Processing*, 2003.

[21] W. Winkler, "Overview of record linkage and current research directions," BUREAU OF THE CENSUS, Tech. Rep., 2006.

[22] D. Jan and L. Zeevaert, *Receptive multilingualism: Linguistic analyses, language policies and didactic concepts*. John Benjamins, 2007, vol. 6.

[23] T. Landauer, P. Foltz, and D. Laham, "An introduction to latent semantic analysis," *Discourse processes*, vol. 25, no. 2-3, pp. 259–284, 1998.

[24] M. Berry, S. Dumais, and G. O'Brien, "Using linear algebra for intelligent information retrieval," *SIAM review*, vol. 37, no. 4, pp. 573–595, 1995.

[25] T. Loughran and B. McDonald, "Textual analysis in accounting and finance: A survey," *Journal of Accounting Research*, vol. 54, no. 4, pp. 1187–1230, 2016.

[26] M. Wall, A. Rechtsteiner, and L. Rocha, "Singular value decomposition and principal component analysis," in *A practical approach to microarray data analysis*. Springer, 2003, pp. 91–109.

[27] E. Boukus and J. Rosenberg, "The information content of fomc minutes," *Available at SSRN 922312*, 2006.

[28] I. Kountouri, E. Manousakis, and A. Tsekrekos, "Latent semantic analysis of corporate social responsibility reports (with an application to hellenic firms)," *International Journal of Disclosure and Governance*, vol. 16, no. 1, pp. 1–19, 2019.

[29] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural info. processing systems*, 2013.

[30] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *conference on empirical methods in natural language processing (EMNLP)*, 2014.

[31] M. Kusner, Y. Sun, N. Kolkin, and K. Weinberger, "From word embeddings to document distances," in *International conference on machine learning*, 2015.

[32] S. Arora, Y. Liang, and T. Ma, "A simple but tough-to-beat baseline for sentence embeddings," 2016.

[33] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *Int. conf. on machine learning*, 2014.

[34] R. Kiros, Y. Zhu, R. Salakhutdinov, R. Zemel, R. Urtasun, A. Torralba, and S. Fidler, "Skip-thought vectors," in *Advances in neural information processing systems*, 2015.

[35] A. Conneau, D. Kiela, H. Schwenk, L. Barrault, and A. Bordes, "Supervised learning of universal sentence representations from natural language inference data," in *Conference on Empirical Methods in Natural Language Processing*, 2017.

[36] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[37] D. Cer, Y. Yang, S. Kong, N. Hua, N. Limtiaco, R. John, N. Constant, M. Guajardo, S. Yuan, C. Tar *et al.*, "Universal sentence encoder," *arXiv preprint arXiv:1803.11175*, 2018.

[38] V. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals." *Soviet Physics Doklady*, vol. 10, no. 8, pp. 707–710, 1966, doklady Akademii Nauk SSSR, V163 No4 845-848 1965.

[39] S. Zhang, Y. Hu, and G. Bian, "Research on string similarity algorithm based on levenshtein distance," in *2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, 2017.

[40] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, p. 131, 2008.

[41] E. Engström and P. Runeson, "Decision support for test management and scope selection in a software product line context," in *Fourth Int. Conf. on Software Testing, Verification and Validation Workshops*, 2011.

[42] Z. Lipton, C. Elkan, and B. Naryanaswamy, "Optimal thresholding of classifiers to maximize f1 measure," in *Machine Learning and Knowledge Discovery in Databases*, 2014.

[43] T. Fawcett, "An introduction to roc analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861 – 874, 2006, rOC Analysis in Pattern Recognition.

[44] S. Tahvili, W. Afzal, M. Saadatmand, M. Bohlin, and S. H. Ameerjan, "Espret: A tool for execution time estimation of manual test cases," *Journal of Systems and Software*, vol. 146, pp. 26 – 41, 2018.

[45] C. Robson, *Real world research : a resource for users of social research methods in applied settings*. Wiley, 2011.

[46] S. Tahvili, L. Hatvani, M. Felderer, W. Afzal, M. Saadatmand, and M. Bohlin, "Cluster-based test scheduling strategies using semantic relationships between test specifications," in *5th Int. Workshop on Requirements Engineering and Testing*, 2018.

[47] S. Tahvili, L. Hatvani, M. Felderer, W. Afzal, and M. Bohlin, "Automated functional dependency detection between test cases using doc2vec and clustering," in *The First IEEE International Conference On Artificial Intelligence Testing*, 2019.

[48] S. Arlt, T. Morciniec, A. Podelski, and S. Wagner, "If a fails, can b still succeed? inferring dependencies between test results in automotive system testing," in *The IEEE 8th International Conference on Software Testing, Verification and Validation*, 2015.

[49] S. Tahvili, M. Saadatmand, S. Larsson, W. Afzal, M. Bohlin, and D. Sudmark, "Dynamic Integration Test Selection Based on Test Case Dependencies," in *The 11th Workshop on Testing: Academia-Industry Collaboration, Practice and Research Techniques*, 2016.

[50] S. Tahvili, M. Ahlberg, E. Fornander, W. Afzal, M. Saadatmand, M. Bohlin, and M. Sarabi, "Functional dependency detection for integration test cases," in *Companion of the 18th IEEE Int. Conf. on Software Quality, Reliability, and Security.*, 2018.

[51] S. Tahvili, R. Pimentel, W. Afzal, M. Ahlberg, E. Fornander, and M. Bohlin, "sortes: A supportive tool for stochastic scheduling of manual integration test cases," *Journal of IEEE Access*, vol. 6, pp. 1–19, 2019.