Performance Comparison of Two Deep Learning Algorithms in Detecting Similarities Between Manual Integration Test Cases

Cristina Landin*‡, Leo Hatvani§, Sahar Tahvili†§, Hugo Haggren†, Martin Längkvist‡, Amy Loutfi‡, Anne Håkansson¶
* Product Development Unit Radio, Production Test Development, Ericsson AB, Kumla, Sweden
cristina.landin@ericsson.com
†Global Artificial Intelligence Accelerator (GAIA), Ericsson AB, Stockholm, Sweden
{sahar.tahvili, hugo.haggren}@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
leo.hatvani@mdh.se
¶School of Electrical Engineering and Computer Science, KTH Royal Institute of Technology, Stockholm, Sweden
annehak@kth.se

***Abstract*—Software testing is still heavily dependent on human judgment since a large portion of testing artifacts, such as requirements and test cases are written in a natural text by experts. Identifying and classifying relevant test cases in large test suites is a challenging and also time-consuming task. Moreover, to optimize the testing process test cases should be distinguished based on their properties, such as their dependencies and similarities. Knowing the mentioned properties at an early stage of the testing process can be utilized for several test optimization purposes, such as test case selection, prioritization, scheduling, and also parallel test execution. In this paper, we apply, evaluate, and compare the performance of two deep learning algorithms to detect the similarities between manual integration test cases. The feasibility of the mentioned algorithms is later examined in a Telecom domain by analyzing the test specifications of five different products in the product development unit at Ericsson AB in Sweden. The empirical evaluation indicates that utilizing deep learning algorithms for finding the similarities between manual integration test cases can lead to outstanding results.**

Keywords–Natural Language Processing; Deep Learning; Software Testing; Semantic Analysis; Test Optimization

I. INTRODUCTION

Employing Artificial Intelligence (AI) techniques for test optimization purposes in the industry is regarded to be beneficial due to their ability to analyze the complex software model and a large amount of generated test data [1]. One of the principal ideas behind test optimization is to test a subset of the test cases (in any form of test case selection, prioritization, and test suite minimization) while still covering the requirements [2]–[5]. Parallel test execution [6] and test execution scheduling [7] can be also considered as promising approaches to optimizing the testing process [8]. However, while employing the aforementioned approaches, manual work, domain knowledge, and human judgment are still required. Due to utilizing AI technologies, such as Natural Language Processing (NLP), machine learning and deep learning can help to reduce human effort and improve the performance of the optimization approaches. Additionally, test optimization has been considered as a multi-criteria optimization problem [3], where several criteria, such as dependency and similarity between test cases, execution time, and requirement coverage are recognized as critical elements for selecting, ranking, scheduling, or removing any test case in the testing cycle. As stated earlier, in manual test execution, test cases are designed and created manually, which may result in having similar test cases that are just designed or written differently. Finding the similarities between test cases opens the possibility to apply the aforementioned optimization approaches, such as parallel test execution. In fact, similar test cases can be clustered together and executed at the same time, in parallel with other test cases. Knowing the similarities between test cases at an early stage of a testing process can help us execute test cases more efficiently and directly, thus reducing costs and time [6] [9]. In this paper, we apply two deep learning algorithms for finding the similarities between manual integration test cases that are designed for testing five different products at Ericsson AB. Later, similar test cases are clustered and proposed for execution in parallel. This paper makes the following contribution:

- Applying and comparing the performance of two deep learning algorithms, finding the similarities between manual integration test cases against labeled data conducted from the so-called subject matter experts (SME).
- Clustering similar test cases and scheduling them for parallel test execution.

Moreover, the proposed approach in this paper is utilized to develop our previous work [6], for the similarity threshold calculation. The organization of this paper is as follows: Section II provides a background and problem statement. Section III presents an overview of research on NLP in the testing domain. Section IV describes the proposed approach in this paper. An industrial case study laid out in Section V. Section VI analyzes the performance between the utilized deep learning algorithms is compared against the labeled data. Threats to validity and delimitations are discussed in Section VII. Section VIII clarifies some points of future directions of the present work and finally, Section IX concludes the paper.

 ISBN: 978-1-61208-827-3

TABLE I. TWO EXAMPLES OF SIMILAR AND NON-SIMILAR TEST CASES. $TC_{i,j}$, WHERE TC STANDS FOR A TEST CASE, i IS THE TEST CASE NUMBER AND j IS THE PRODUCT NAME. THE HIGHLIGHTED WORDS ARE IMPORTANT WORDS FOR COMPARISON BETWEEN TEXTS.

Annotation	Test case specification
Similar	$TC_{1,A}$: This test case will **measure** the **current** value of **LED 1**. Start by **supplying 2.4 V** into **Pin** 2 to the FPGA G1 and **read** the output of **Pin 5**. **Pass** criteria is **10 mA**. **Save** the **result** in **database** 1. $TC_{2,B}$: **Measure** the **current** across **LED 2**. **Supply 2.4 V** into **Pin 2** to the FPGA G2 and **read** the output of **Pin 5**. **Pass** criteria is **15 mA**. **Save** the **result** in **db** 2.
Non-Similar	$TC_{1,A}$: This test case will **measure** the **current** value of **LED 1**. Start by **supplying 2.4 V** into **Pin** 2 to the FPGA G1 and **read** the output of **Pin 5**. **Pass** criteria is **10 mA**. **Save** the **result** in **database** 1. $TC_{3,B}$: This test case will **measure** the **voltage** of **LED 1**. **Supply 10 V** to the **input T1** and **measure** the **voltage** in **T2**. **Compare** the **results** with the **calculated** and **save** the **result** in **database** 1.

II. BACKGROUND

Ericsson AB is one of the leading providers of Information and Communication Technology (ICT) and has, among many units, a business unit network, which produces the latest technology in Radio Base Stations (RBS). An RBS is a radio transceiver used in wireless communication. Figure 1 shows a block diagram of an RBS, where the transmitter and receiver are the radio parts and the digital control supplies and receives the digital signals to both transmitter and receiver.

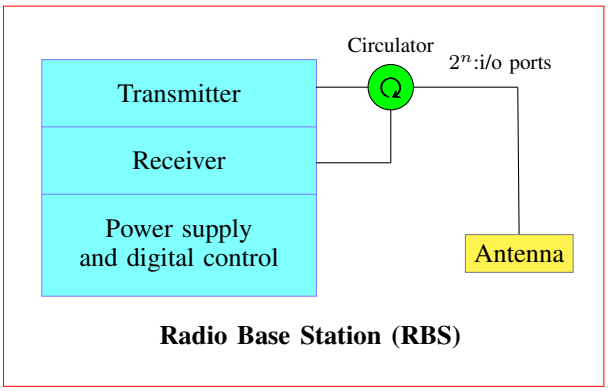


Figure 1. A block diagram of a simplified version of a Radio Base Station 2^n is the number of ports.

The power supply powers the RBS and its specification depends on the target market (e.g., USA, Europe, and Asia). Because both receiver and transmitter contain analog devices, these parts must be calibrated and tested at the device level and unit level together with the antenna. The number of ports to be tested in production is $2^n, (n = 1, 2, 3, ...)$ and each port may be tested separately. A circulator is a simplified version of the design, which only allows the signal to go one direction and not vice versa to protect the internal components. Among all faced challenges the new generation of RBSs are bringing into test and production, the test capacity and the long run-time of each test process can be considered as the main two challenges. The increase in functionality and the number of features (e.g., emerging technologies) that the new generation of products needs to cover aggravate the complexity of the RBS design. Although the design is more complex, the RBS still has to be tested to follow international standards, e.g., 3GPP, FCC, IEC, UL. These standards specify the technical protocols and requirements for mobile communication systems. The above-mentioned factors have resulted in the more complex testing process which increases the required number of test cases that need to be tested. Executing all generated test cases for testing an RBS without any specific order could lead to the waste of testing resources and time. Equation (1) indicates the total testing time, T, when the test cases are executed sequentially. Because of the RBS complex design, such as the number of ports, 2^n, and the number of technologies to cover, h, the total test time increases exponentially and linearly respectively. t_i represents the test time to run each test case and m is the total number of test cases.

$$T = h * 2^n * \sum_{i=1}^{m} t_i \quad (1)$$

Hereof parallel test execution has been considered as a potential solution to optimize the integration testing process at Ericsson. Moreover, the saving time (ST) using parallel test execution for similar test cases can be calculated as:

$$ST = \frac{(np-1)}{np} \times 100 \quad (2)$$

where np is the number of test cases with the same system setup. We define the concept of two similar test cases in this paper as:

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

In other words, if two or more test cases are semantically similar, they might be designed to test the same functionality or require the same system setup or pre-condition. Examples of similar and non-similar test cases are shown in Table I, where LED is Light Emitting Diode and FPGA stands for Field-Programmable Gate Array. They give a glimpse of how the test cases are described in natural language and their pre-requisites. Although parallel test execution is a promising approach for time and resource-saving, however, there are some assumptions that need to be satisfied. For instance, there are no dependencies between test cases and there are enough available test stations to execute test cases simultaneously. In fact, utilizing (2) (in one example scenario) means that the system setup or installation effort needs to be performed once for product A where $np - 1$ products can be executed after product A on the same test station. Since the designed test cases for testing RBSs are written in a non-formal language, employing NLP techniques for semantic analysis might provide some clues to detecting the similarities between test cases. Furthermore, similar test cases can be grouped into several clusters based on their semantic text similarities. On the other hand, finding the similarities between text cases manually requires human work where the testers need to go through the test case descriptions and comparing

 ISBN: 978-1-61208-827-3

them with each other. However, analyzing a large number of test cases manually is a time-consuming process and suffers from ambiguity and uncertainty. Consequently, employing AI algorithms with a combination of human supervision can be considered as a promising approach, where the knowledge of the testers can be captured during the learning process. In addition, the use of NLP techniques can help test engineers to extract relevant information in a large document automatically and thereby distinguishing test cases from each other. Usually, test engineers use different terminologies and the quality of a test case specification is dependent on the test engineers' knowledge, experience, and skill. Therefore, it is crucial to find a proper NLP algorithm to meet the goal. In this regard, we decide to compare the performance of two deep learning algorithms to detect the similarities between test cases and comparing the performance of them against the labeled data conducted by SMEs at Ericsson. Deep learning algorithms for NLP have a high tolerance to noisy data and they are able to classify patterns on which they have not been trained [10], [11].

III. Related Work

There have been several techniques proposed for the crucial step of test optimization. The proposed strategies vary depending on the industrial product, development method, complexity of the models, and data availability and reliability [12]. The overall aim of test optimization is to minimize the required testing time by detecting failed test cases as early as possible. Since the duration and relevancy vary between test cases, the testing time can be reduced by test case selection, prioritization, minimization, and test execution scheduling. The proposed test case prioritization by Nardo et al. [13] uses a coverage-based method that prioritizes test cases with the most recent code changes. This strategy is suitable for software development practice, such as Continuous Integration (CI) [14] [15] [9] where the components are regularly implemented and tested during the development process. Fowler and Foemmel [16] propose an information retrieval approach for detecting code changes that are used for regression test prioritization. For full system testing [17] or when tracking of code changes is unavailable, history-based test prioritization has been proposed by several researchers [18], [19] for prioritizing test cases that have failed more frequently in the past. Spieker et al. [20] propose an adaptive reinforcement learning approach for history-based test case prioritization and selection that learns to rank test cases based on their duration, previous last execution, and failure history. One of the disadvantages of learning-based approaches is that it requires a history of executed test cases and the learning process needs to be repeated for any new product. Test case selection and prioritization based on human-designed test cases have the advantage of evaluating the test cases before they are even executed. Previously [6] used the Levenshtein distance to detect the similarity between each pair of test cases (which were designed to test five different products at Ericsson AB) to detect the similarities between them. The test case pairs were clustered into four groups (identical, very similar, similar, and partially similar) ranked for parallel test execution. The threshold for each of the mentioned groups in [6] is assigned manually using the subject matter expert (SME) experience. The NLP technique that was used in [6] did not take into account the order of the words in a test specification, which can be a disadvantage, considering how specific a procedure of each test case must be implemented. Using NLP techniques has received a great deal of attention in different domains, such as social network analysis [21]. NLP has also been studied to find sentiment analysis [22] and to find semantic and syntactic similarities in large context [23], where the training complexity can be considered as an important aspect. Moreover, using deep learning for natural language analysis has been a focus since the year 2000. Young et.al [24] discuss the recent trends in deep learning on NLP tasks. For learning word representations, Le and Mikolov [25] introduced a new method called Word2vec that finds semantic similarities between paragraphs taking into account the order and semantic context of the vectors (not only limited to a sentence or a fixed length of text), in order to predict words based on the content of the paragraphs. Patil et al. [26] use the Word2vec algorithm as inputs to the Convolutional Neural Network (CNN) to classify binary and multi-class document categorization. Although CNN is effective in finding semantic similarities, it has problems to preserve sequential order and model long-distance dependencies. Moreover, Tahvili et al. [27] [28] employed the Doc2vec algorithm (which is an extension of Word2vec) for finding the functional dependencies between manual integration test cases at Bombardier transportation.

In the present study, we aim to compare the performance of two well-known deep learning algorithms, Doc2vec and Sentence Bidirectional Encoder Representations from Transformers (SBERT), in a set of labeled data from five RBSs from Ericsson AB. Moreover, the similarity threshold which has been assumed manually in [6] can now be automatically computed though comparing the utilized edit distance approach with the deep learning algorithms.

IV. Proposed Approach

The proposed approach in this paper is mirrored in Figure 2. The manual test cases are the input to our model. In step 1 (**Similarity Analysis**) in Figure 2, the similarities between test cases are detected by employing deep learning algorithms, such as Doc2vec and SBERT. Applying deep learning models for semantic analysis provides a set of high dimensional vectors, where each vector represents a test case. To split test cases (step 2 in Figure 2) a clustering algorithm needs to be utilized which can handle high dimensional data points. Finally, since the main application of the proposed approach in Figure 2 is to test efficiency, similar test cases can be scheduled for test execution, e.g., parallel test execution. We need to consider that, the proposed approach in Figure 2 is not just limited to the mentioned algorithms, where other approaches, such as edit distance, string matching, and text classification can be utilized as well. Later in this paper, we provide more information regarding employing other techniques for both **Similarity Analysis** and **Splitting Test Cases** steps in Figure 2. Furthermore, in the upcoming subsections, more details are provided for the utilized algorithms for each step.

A. Doc2vec

Deep learning algorithms use representation learning to learn the data representation instead of using manually hand-crafted features [29]. Doc2vec is a deep neural networks-based

 ISBN: 978-1-61208-827-3

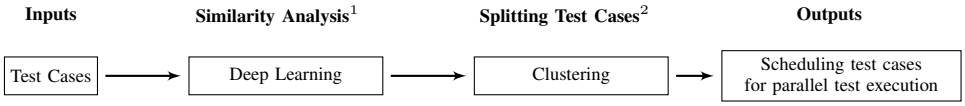


Figure 2. The block diagram of the proposed approach.

algorithm that generates a vector representation for a word, paragraph, or document [25]. Doc2vec represents a non-fixed length document into a vector. Besides, it concatenates each word of the document. Figure 3 shows a representation of the structure of the Doc2vec algorithm to predict the next word. For instance, $W3$ based on the sentence $[W1{+}W2]$. Producing in this way two vectors, one vector for the document, called D and one vector for the words called W. One important facility of this algorithm is to keep the words' properties and the relations between words and semantics of the whole document.

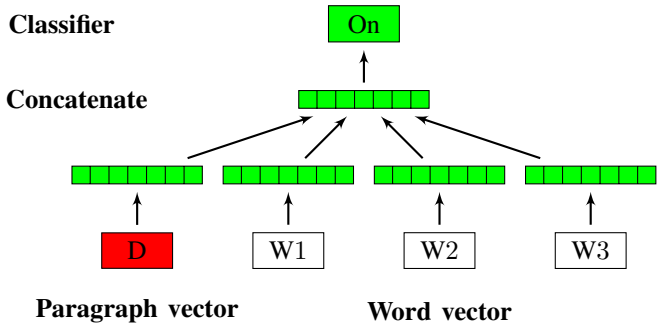


Figure 3. A representation of the Doc2vec algorithm to predict a word, based on the semantics of the sentence.

In fact, each paragraph will have an ID represented in paragraph vector D, and the words are represented in the word vector W. They will concatenate to predict the next word of the sentence.

B. SBERT

SBERT is an algorithm in the domain of NLP, which is designed to pre-train deep bidirectional representations from an unlabeled text [30]. SBERT is a modified version of BERT [31] that jointly conditions on both left and right context in all the layers. Reimers and Gurevych [32] developed sentence embedding based on Euclidian distance which allows finding semantic similarities in sentences and is suitable for clustering and information retrieval purposes. Furthermore, SBERT is computationally more efficient than the heavy and complex BERT. Figure 4 shows the structure of SBERT to compute similarity scores. SBERT adds a pooling operation to the output of BERT and computes the cosine-similarity between sentence embeddings u and v. The pooling operation is to derive a fixed-sized sentence embedding.

The pooling operation is added after BERT to give the same size to the sentence embeddings, u, and v. After this step, the similarities are computed using cosine-distance.

C. HDBSCAN

For the clustering part of the proposed approach in this paper, we propose to use the Hierarchical Density-Based Spatial

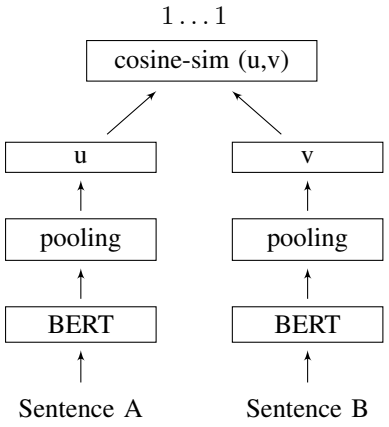


Figure 4. The architecture of the SBERT algorithm which is designed to compute similarity scores between sentences.

Clustering of Applications with Noise (HDBSCAN), which is a density-based clustering algorithm based on a hierarchical density estimate [33]. HDBSCAN generates a simplified hierarchy and extracts the most significant clusters. Using HDBSCAN has two main advantages: 1-HDBSCAN is able to clusters the high dimensional data point without employing any dimension reduction technique (e.g., Principal Component Analysis (PCA)), 2-HDBSCAN provides a cluster of non-clusterable data points which can be interpreted as noises, outliers or anomalies. In this paper, the non-clusterable data points are considered as non-similar test cases. However, if the high dimensional data is projected onto a lower dimension space (using a technique like PCA), other standard clustering techniques can be employed instead of HDBSCAN.

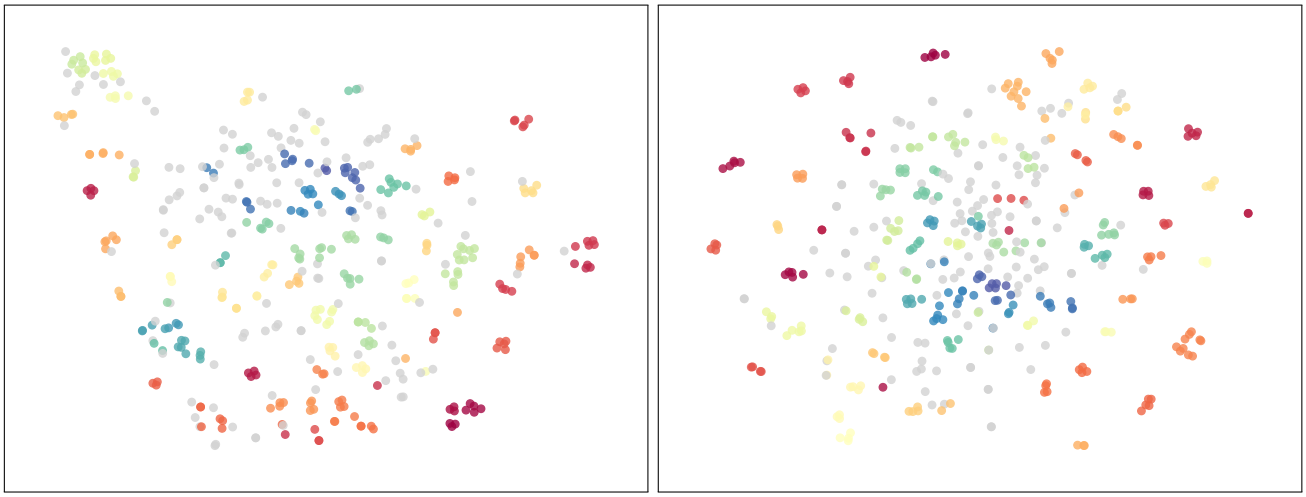
V. Industrial Case Study

The provided industrial case study in this work follows the proposed guidelines for conducting and reporting case study research in software engineering by Runeson and Höst [34] and specifically the way guidelines are followed in [35] and [3], [36]. Hereof, five multi-standard RBSs compatible with their respective number of test cases are utilized as a case under study.

A. Unit of analysis and procedure

The test specifications of five products of the 4th Generation (4G) RBS are our dataset, which includes 444 test cases in total. The utilized test cases are written in natural language text and by the SME at Ericsson. As illustrated in Figure 2 our approach aims to cluster similar test cases and also provide a cluster of non-similar test cases. Table I provides two real industrial examples of similar and non-similar test cases. However, using human knowledge and judgment for the similarity analysis is a time and resource-consuming process

 ISBN: 978-1-61208-827-3



(a) The clustered test case using the generated vectors by the SBERT. (b) The clustered test case using the generated vectors by the Doc2vec.

Figure 5. The clustered test cases using HDBSCAN algorithm on the generated vectors by two deep learning algorithms. There are 75 and 76 clusters plotted in different colors for SBERT and Doc2vec respectively. The outliers are shown in gray.

and it might suffer from ambiguity and uncertainty. Therefore, employing AI techniques for similarity analysis and thereby clustering purposes is critical in large industries.

B. Experimental Setup

SBERT provides the option to train a new model or to use a pre-trained model to convert test cases to feature vectors. As our dataset is rather small and built out of test cases written in English, we are able to use a pre-made BERT-base model with mean-tokens pooling, *bert-base-nli-mean-tokens*, obtainable from the SBERT repository [37]. We have not made any customizations to the code and thus the result is a feature vector with 768 features for each of the 444 observed test cases. The employed implementation of the SBERT produces consistent results between different runs. For the Doc2vec approach to feature vector generation, we have used the Gensim [38] implementation. For this approach, we have used the following parameters: feature vector size of 100, min word count for dictionary inclusion of 1, and 100 training epochs. Gensim approach is based on training a new neural network every iteration and thus can produce various results depending on the input parameters and a random initial value. To estimate the impact of these parameters, we varied them within reasonable ranges across 150 iterations and found that the largest standard deviation from the results presented in this paper is 0.08 across all of the measures presented in Table III.

The generated feature vectors are then clustered using the standard implementation of the HDBSCAN algorithm using the default clustering values with distances being computed as cosine distances between feature vectors. Pairs of test cases from the labeled data are then compared against the clusters to obtain the confusion matrix from which precision, recall, and F1 score are calculated. Further details about the implementation of the HDBSCAN algorithm are available at [39].

VI. Result

Figure 5 shows the clustered test cases using the HDBSCAN, each color represents a unique cluster of test cases which are being considered as semantic similar by Doc2vec and SBERT algorithms. Figure 5a illustrates the obtained clusters using the provided vectors by SBERT, where the number of obtained clusters is equal to 75. Figure 5b shows the clusters where HDBSCAN used the generated vectors provided by Doc2vec, the number of obtained clusters is equal to 76. The t-SNE is used to plot the results after the clustering done by HDBSCAN. Both results are expected due to the properties of the dataset and resemble the SME's criteria. We need to consider that the size of each cluster is different for the Doc2vec and SBERT. Moreover, using HDBSCAN can help us to have a cluster of non-clusterable data points, which indicates to non-similar test cases in this study.

A. Model Performance Evaluation

Table II summarizes the cluster size and the obtained non-clusterable data points. The entire results and implementation source are available at [40]. In order to compare the performance of the employed deep learning algorithms against the labeled data, 402 out of 444 test cases manually labeled by the SME at Ericsson, wherein in total we received labels for 211 similar and 191 non-similar test cases. Test cases similarity detection might suffer the class imbalance problem when the class distributions (similar and non-similar) are imbalanced. Therefore, selecting a suitable performance metrics is critical and also influences the measured performance of a model [41]. To evaluate the performance of the proposed approach in this paper, precision, recall, and F1 score are measured instead of accuracy. F1 score conveys the balance between the precision and the recall, which is a suitable metric for the imbalance problems, shown in (3).

$$F1 \quad score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{3}$$

 ISBN: 978-1-61208-827-3

TABLE II. THE NUMBER OF CLUSTERS, CLUSTER SIZE, AND THE NUMBER OF NON-CLUSTERABLE DATA POINTS USING DOC2VEC, SBERT, AND HDBSCAN.

	Cluster Size	
Cluster Number	Doc2vec	SBERT
Cluster 1	5	3
Cluster 2	5	5
Cluster 3	7	3
⋮		
Cluster 75	3	2
Cluster 76	5	0
Non-clusterable	132	133

Table III summarizes the evaluation of the obtained results, using precision, recall, and F1 score.

TABLE III. THE PERFORMANCE COMPARISON OF DOC2VEC AND SBERT AGAINST THE LABELED DATA PROVIDED BY THE SME.

	Similar			Non-similar		
Algorithm	Precision	Recall	F1 score	Precision	Recall	F1 score
Doc2vec	0.952	0.943	**0.947**	0.937	0.947	**0.942**
SBERT	0.946	0.834	0.886	0.837	0.947	0.889

As can be seen in Table III, the F1 score for Doc2vec is 0.947 and 0.942 for similar and non-similar, respectively. The obtained F1 score for SBERT is about 0.89, which indicates a good overall performance as well but not as good as that of Doc2vec. Moreover, the presented clusters in Figure 5 and Table II can be utilized for parallel test execution and also test suite reduction. For parallel test execution, each cluster (which contains several test cases) can be scheduled for execution simultaneously. As mentioned before, if two test cases are similar in their test specifications they might be designed for testing the same functionality or they required the same precondition, i.e. in system setup. Using the presented results in Table II for instance for the Doc2vec algorithm, we have 5 test cases (distributed between five different products) which are grouped into cluster 1. There are two different scenarios in this case. 1-The mentioned five test cases are designed to test a common function between five different products. In this scenario, executing all five mentioned test cases leads to this function will be tested fully before we are scheduling another function for the testing. All other presented clusters in Table II can be executed parallel on the other test stations, while cluster 1 is executed completely. 2-The mentioned five test cases in cluster 1 are required the same installation effort. Thus the testing environment needs to be configured once for just one test case inside of cluster 1 and all other four test cases can be executed after each other on the same test station. Using 2 can help us to measure the saving time for this scenario. On the other hand, selecting just one test candidate from each cluster for execution can be utilized for the test suite reduction purposes. Furthermore, the non-clusterable data points presented in Table II indicate the non-similar test cases which can be executed sequentially before or after other similar test cases.

B. The Similarity Threshold Calculation

Previously [6], the test cases were classified into several classes (e.g., identical, very similar, similar) based on their Levenshtein distance and the similarity threshold, which was set manually using the SME's experience. In this regard, the similarity threshold is selected as $0.8 \leqslant \text{LD} \leqslant 1$, where 0.8 is the desired lower limit for the similarity of two test cases. Moreover, a Levenshtein distance equal to 1 represents two identical test cases and a Levenshtein distance lower than 0.8 represents non-similar test cases. Finding an optimal similarity threshold is beneficial in terms of reducing human judgment. It can also be utilized for identifying the first distance that test cases are started to be similar to each other. In this study, an automatic threshold detection approach (see **Threshold Detection**$^{2\prime}$ in Figure 6) is applied through measuring the Levenshtein distance between the vectors (which belong to the same cluster) generated by the deep learning models. The second contribution of this paper is to find the similarity threshold to delimit between similar and non-similar test cases using The Levenshtein distance in our previous work [6]. The Levenshtein distance is measured between all test cases that have been clustered as similar by HDBSCAN. Table IV presents The sum of the averages Levenshtein distance per cluster. In fact, the Levenshtein distance between each test case, which ended up into the same clustered, is measured. As can be seen, the mean value of the Levenshtein distances is equal to 0.69 and 0.64 for Doc2vec and SBERT respectively.

TABLE IV. THE SIMILARITY THRESHOLD CALCULATION USING DEEP LEARNING ALGORITHMS (DOC2VEC AND SBERT) AND CLUSTERED RESULTS (HDBSCAN) BASED ON LEVENSHTEIN DISTANCE.

Algorithm	Doc2vec	SBERT
Number of clusters	76	75
The sum of the averages Levenshtein distance per cluster	**0.69**	**0.64**

This result indicates that test cases can already be considered similar if their Levenshtein distance is equal or greater than 0.64 for SBERT and 0.68 for Doc2vec respectively. This improves the results found in [6], where the lowest boundary of similarity was assumed 0.8 in close consultation with SMEs at Ericsson. Moreover, these performance measures in this study are an improvement to the edit distance approach which used by us previously [6]. The F1 score of 0.61 using Levenshtein distance obtained utilizing the same dataset. According to the presented result in Table III, Doc2vec has a better performance compared to both SBERT and significantly better than Levenshtein distance on this application.

VII. THREATS TO VALIDITY

There are some risks in applying the proposed approaches in this paper. Infrastructure, enough available testing stations, and resource limitation can be considered as the major construct validity threat to the parallel test execution. For applying the proposed approach in this study, 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. Using the test specifications for finding the similarities between test

 ISBN: 978-1-61208-827-3

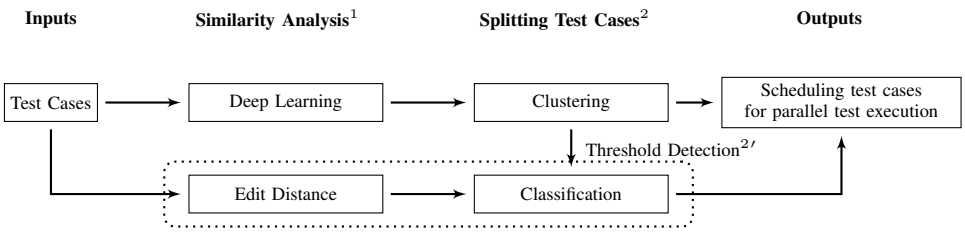


Figure 6. The similarity threshold calculation for the Edit distance approach using the clustering results. The lower part of the diagram shows the approach used previously in [6].

cases might be sensitive in terms of changing a single character in the test. For instance, $TC_{1,A}$ in Table I, the change of only one character in the pass criteria from $10mA$ to $100mA$ could affect much in the measurement results of a LED, which could result in a broken component. However, deep learning algorithms are more stable to the character changing compare to the edit distance and string matching approaches. The proposed approach in this paper has been applied on just one industrial testing project in the Telecom domain, however, it should be applicable to other similar contexts in other testing domains. Nevertheless, we cannot claim that the proposed approach in this paper works well in all fields where precision is a very important variable, e.g., medicine, chemistry, and electronics. Furthermore, the labeled data, which has been used for the performance evaluation was conducted manually by the testing expert at Ericsson and it might suffer from uncertainty. Therefore, another type of ground truth needs to be conducted in order to generalize the proposed approach in this study. Finally despite deep learning and word embedding approaches being interesting, one cannot exclude that traditional rule-based methods may also be applicable and sometimes result in better performance than the latest deep learning methods, which depend on the application. In our case, rules are given by the specialist in the area which could give more importance to specific sections on the test case description to find better insights and extract relevant features before clustering.

VIII. Discussion and Future Work

The main goal of this study is to compare the performance of two deep learning algorithms on an industrial case study. To this end, we make the following contributions: 1-Doc2vec and SBERT have been employed to detect the similarities between manual integration test cases automatically. The similarities have been extracted by analyzing test case specifications written in a non-formal natural language. 2-The evaluation of the proposed algorithms was performed by conducting an industrial testing project in the Telecom domain at Ericsson in Sweden. 3-The performance of the Doc2vec and SBERT was compared against the labeled data using SME knowledge at Ericsson. The obtained results show the outstanding performance of the mentioned deep learning algorithms on the conducted industrial case study. 4-The performance of the Doc2vec and SBERT was compared against our previous work [6] where the Levenshtein distance was utilized for the similarity detection on the same dataset. The obtained results indicate that the lowest boundary of similarity using Levenshtein distance can go down to 0.64 compared to 0.8, which was the empirical value used previously in [6] by the SME.

A. Future Work

The main future direction of this paper is employing other text analysis approaches such as edit distance and string matching for similarity detection. In fact, the mentioned approaches in Figure 6 can be extended to all existing text mining methods. Other clustering and classification algorithms can be adopted to Figure 6. As stated earlier, using data dimensionality reduction techniques, e.g., random forests, PCA and thereby applying other standard clustering techniques (e.g., k-means) might provide better results compared to HDBSCAN. Moreover, conducting a larger case study and comparing the performance of all text mining methods, which are applicable for similarity analysis, can provide a clue for finding the best method in terms of accuracy and execution time. Generally, running a deep learning or neural network algorithm requires more time compared to the other text-mining algorithm, which has a less complex structure. Developing the utilized algorithms of this study as a tool that can handle even larger sets of test specifications is one of the future directions. Despite the fact that the results found in this paper using deep learning algorithms are promising, they are entirely based on one dataset within a specific domain i.e. five fourth-generation (4G) RBSs. We aim to test this approach in other datasets and domains thus, in this way it can be generalized and transfer to other products, e.g., fifth-generation (5G) RBSs or even future generations of RBSs. Furthermore, we are aiming to verify the robustness of the tool. For this purpose, a long time study is needed within production, which will secure this tool to be scalable and compatible with different platforms proper of old and new technologies. A question left to answer is whether these deep learning approaches are better than the traditional rule-based methods for this application. Usually, the traditional rule-based methods are hand-crafted and require field knowledge, which may not be a disadvantage in very complicated industrial applications. Although we can see many potential improvements to the traditional rule-based methods, a formal comparison of those methods and deep learning methods is worth investigating.

IX. Conclusion

Parallel test execution plays a vital role in test optimization and can lead to saving time and cost in a testing process. In this paper, two deep learning algorithms (Doc2vec and SBERT) are applied and evaluated to find the semantic similarities between manual integration test cases for test optimization. The obtained results indicate that Doc2vec shows better performance (F1 score=0.947 for the similar test cases and F1 score=0.942 for non-similar test cases) compare to SBERT (F1 score=0.886

 ISBN: 978-1-61208-827-3

for the similar test cases and F1 score=0.889) when it evaluated against the labeled data. This conclusion opens possibilities to use the method for parallel testing and test suite minimization.

ACKNOWLEDGMENT

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

REFERENCES

[1] S. Khan and T. Yairi, "A review on the application of deep learning in system health management," *Mechanical Systems and Signal Processing*, vol. 107, pp. 241–265, 2018.

[2] R. Ducloux, L. Fourment, S. Marie, and D. Monnereau, "Automatic optimization techniques applied to a large range of industrial test cases," *Int. Journal of Material Forming*, vol. 3, no. 1, pp. 53–56, 2010.

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

[4] S. Tahvili et al., "Dynamic Integration Test Selection Based on Test Case Dependencies," in *The 11th Workshop on Testing: Academia-Industry Collaboration, Practice and Research Techniques*, 2016.

[5] S. Tahvili et al., "Towards earlier fault detection by value-driven prioritization of test cases using fuzzy topsis," in *The 13th Int. Conf. on Information Technology: New Generations*, 2016, pp. 745–759.

[6] C. Landin et al., "Cluster-based parallel testing using semantic analysis," in *2020 IEEE International Conference on Artificial Intelligence Testing (AITest)*, 2020, pp. 99–106.

[7] S. Tahvili et al., "sortes: A supportive tool for stochastic scheduling of manual integration test cases," *Journal of IEEE Access*, pp. 1–19, 2019.

[8] G. Rothermel, R. Untch, C. Chu, and M. Harrold, "Test case prioritization: An empirical study," in *Proceedings IEEE International Conference on Software Maintenance-1999 (ICSM'99).'Software Maintenance for Business Change'(Cat. No. 99CB36360)*. IEEE, 1999, pp. 179–188.

[9] S. Tahvili et al., "Cost-benefit analysis of using dependency knowledge at integration testing," in *The 17th Int. Conf. On Product-Focused Software Process Improvement*, 2016, pp. 268–284.

[10] T. Khuat and B. Gabrys, "A comparative study of general fuzzy min-max neural networks for pattern classification problems," *Neurocomputing*, vol. 386, pp. 110 – 125, 2020.

[11] O. Engström, S. Tahvili, A. Muhammad, F. Yaghoubi, and L. Pellaco, "Performance analysis of deep anomaly detection algorithms for commercial microwave link attenuation," in *The 2020 International Conference on Advanced Computer Science and Information Systems*, October 2020.

[12] A. Petrenko, A. Dury, S. Ramesh, and S. Mohalik, "A method and tool for test optimization for automotive controllers," in *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*, 2013, pp. 198–207.

[13] D. Nardo, N. Alshahwan, L. Briand, and Y. Labiche, "Coverage-based regression test case selection, minimization and prioritization: A case study on an industrial system," *Software Testing, Verification and Reliability*, vol. 25, no. 4, pp. 371–396, 2015.

[14] P. Duvall, S. Matyas, and A. Glover, *Continuous integration: improving software quality and reducing risk*. Pearson Education, 2007.

[15] M. Fowler and M. Foemmel, "Continuous integration," 2006, [Online]. Available from: https://martinfowler.com/articles/continuousIntegration/ 2020.09.02.

[16] R. Saha, L. Zhang, S. Khurshid, and D. Perry, "An information retrieval approach for regression test prioritization based on program changes," in *IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1, 2015, pp. 268–279.

[17] A. Dias, R. Subramanyan, M. Vieira, and G. Travassos, "A survey on model-based testing approaches: a systematic review," in *Proceedings of the int. work. on Empirical assessment of software engineering languages and technologies*, 2007, p. 31–36.

[18] J. Kim and A. Porter, "A history-based test prioritization technique for regression testing in resource constrained environments," in *Proceedings of Int. Conf. on Software Engineering*, 2002, pp. 119–129.

[19] D. Marijan, A. Gotlieb, and S. Sen, "Test case prioritization for continuous regression testing: An industrial case study," in *Int. Conf. on Software Maintenance*, 2013.

[20] H. Spieker, A. Gotlieb, D. Marijan, and M. Mossige, "Reinforcement learning for automatic test case prioritization and selection in continuous integration," in *Proceedings of Int. Symp. on Software Testing and Analysis*, 2017.

[21] A. Sarlan, C. Nadam, and S. Basri, "Twitter sentiment analysis," in *Int. conf. on Information Technology and Multimedia*, 2014.

[22] P. Sanguansat, "Paragraph2vec-based sentiment analysis on social media for business in thailand," in *Int. Conf. on Knowledge and Smart Technology (KST)*, 2016.

[23] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013.

[24] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," 2017.

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

[26] S. Patil, A. Gune, and M. Nene, "Convolutional neural networks for text categorization with latent semantic analysis," in *Int. Conf. on Energy, Communication, Data Analytics and Soft Computing*, 2017.

[27] S. Tahvili, L. Hatvani, M. Felderer, W. Afzal, and M. Bohlin, "Automated functional dependency detection between test cases using doc2vec and clustering," in *Int. Conf. On Artificial Intelligence Testing*, 2019.

[28] S. Tahvili et al., "Cluster-based test scheduling strategies using semantic relationships between test specifications," in *Int. Work. on Requirements Engineering and Testing*, 2018.

[29] D. Erhan et al., "Why does unsupervised pre-training help deep learning?" *J. Mach. Learn. Res.*, vol. 11, p. 625–660, 2010.

[30] J. Lee et al., "BioBERT: a pre-trained biomedical language representation model for biomedical text mining," *Bioinformatics*, no. 4, pp. 1234–1240, 2019.

[31] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2019.

[32] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," in *Conf. on Empirical Methods in Natural Language Processing*, 2019.

[33] R. Campello, D. Moulavi, and J. Sander, "Density-based clustering based on hierarchical density estimates," in *Advances in Knowledge Discovery and Data Mining*, 2013.

[34] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, 2009.

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

[36] S. Tahvili, L. Hatvani, E. Ramentol, R. Pimentel, W. Afzal, and F. Herrera, "A novel methodology to classify test cases using natural language processing and imbalanced learning," *Engineering Applications of Artificial Intelligence*, vol. 95, pp. 1–13, August 2020.

[37] "Sbert source code and model repository," [Online]. Available from: http://github.com/UKPLab/sentence-transformers/ 2020.09.01.

[38] R. Rehurek and P. Sojka, "Software Framework for Topic Modelling with Large Corpora," in *LREC Work. on New Challenges for NLP Frameworks*, 2010.

[39] "Hdbscan source code and model repository," [Online]. Available from github.com/scikit-learn-contrib/hdbscan/ 2020.09.01.

[40] "Model performance evaluation," [Online]. Available from: http://github.com/leohatvani/landin-performance-comparison/ 2020.09.01.

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

 ISBN: 978-1-61208-827-3