Automated Functional Dependency Detection Between Test Cases Using Doc2Vec and Clustering

Sahar Tahvili
RISE SICS Västerås AB
Västerås, Sweden
sahar.tahvili@ri.se

Leo Hatvani
Mälardalen University
Västerås, Sweden
leo.hatvani@mdh.se

Michael Felderer
University of Innsbruck
Innsbruck, Austria
michael.felderer@uibk.ac.at

Wasif Afzal
Mälardalen University
Västerås, Sweden
wasif.afzal@mdh.se

Markus Bohlin
RISE SICS Västerås AB
Västerås, Sweden
markus.bohlin@ri.se

***Abstract*—Knowing about dependencies and similarities between test cases is beneficial for prioritizing them for cost-effective test execution. This holds especially true for the time consuming, manual execution of integration test cases written in natural language. Test case dependencies are typically derived from requirements and design artifacts. However, such artifacts are not always available, and the derivation process can be very time-consuming. In this paper, we propose, apply and evaluate a novel approach that derives test cases' similarities and functional dependencies directly from the test specification documents written in natural language, without requiring any other data source. Our approach uses an implementation of Doc2Vec algorithm to detect text-semantic similarities between test cases and then groups them using two clustering algorithms HDBSCAN and FCM. The correlation between test case text-semantic similarities and their functional dependencies is evaluated in the context of an on-board train control system from Bombardier Transportation AB in Sweden. For this system, the dependencies between the test cases were previously derived and are compared to the results of our approach. The results show that of the two evaluated clustering algorithms, HDBSCAN has better performance than FCM or a dummy classifier. The classification methods' results are of reasonable quality and especially useful from an industrial point of view. Finally, performing a random undersampling approach to correct the imbalanced data distribution results in an F1 score of up to 75% when applying the HDBSCAN clustering algorithm.**

***Index Terms*—Software Testing, Paragraph Vectors, Test Case Dependency, Clustering, Doc2Vec, HDBSCAN, FCM**

I. Introduction

Software testing phase has been a target of optimization in several studies and several models for the test process improvement have been proposed [1]. Most recently, new techniques in the areas of deep learning, machine learning and natural language processing (NLP), as well as the combination of them, have begun to impact agility in software development life cycle (SDLC) phases. According to our experience, several large enterprises, especially in the embedded systems domain, still struggle to increase test productivity and reduce testing costs [2]. There is a strong need to improve quality while at the same time reduce costs, driven by the fast pace of technology change and increased market competition. Software testing phase is naturally the target of many optimization efforts. In our continued collaboration with Bombardier Transportation AB (BT) in Sweden, a large enterprise working in the railway transportation domain, we previously identified the problem of test optimization as a multi-criteria decision making problem, where a set of criteria such as dependencies and similarities between test cases, requirement coverage and test cases' execution time need to be satisfied for minimizing testing cost and maximizing testing coverage [2]–[4]. Our special focus has been on several levels of system integration (referred to as integration testing for simplicity purpose) where the interaction between two (or more) software modules is tested and the modules increasingly interdepend on input/output signals through different interfaces for meeting different functionalities. Consequently, the resulting test cases are more complex (as compared to unit tests) and can be dependent on each other to test the inter-module dependencies which are essential for integration testing. This increased test complexity is not helped by manually executing integration tests, as is the case in our context. In order to automatically measure the impact of one of the mentioned criteria on manual integration test cases (i.e. dependencies between test cases), this paper adopts a set of new techniques inspired from NLP and deep learning. A manual integration test case textually describes what functionality needs to be tested and how. It usually consists of a set of pre-conditions, initial state, testing procedure, and post-conditions. Sometimes in the integration testing, several test cases are designed to test one function and sometimes the interaction between two (or more) functions can be tested by just one test case. Moreover, some test cases require the same system setup, installation and initial states. Since all mentioned elements (pre- and post-conditions, testing procedure, system setup, etc.) are described textually in a manual test case specification, performing NLP-based approaches can help us recognize similar test cases. Since manual test cases are written in natural text, the concept of *semantic similarity* [5] will be used in this paper. However, determining the semantic similarity between test cases is a challenging task, due to the variability of natural language expressions, where the semantic similarity can be measured at document level, term level or sentence level. Several approaches have been proposed for determining the semantic similarity such as edge-counting measures [6], information content-based measures [5], hybrid measures [7] and also feature-based measures [8].

There are several methods proposed for dependency detection on test cases (see e.g. [9] and [3]) but most of them are

ECSEL & VINNOVA (through projects MegaM@RT2 & TESTOMAT) and the Swedish Knowledge Foundation (through the projects TOCSYC (20130085) and TestMine (20160139)) have supported this work.

978-1-7281-0492-8/19/$31.00 ©2019 IEEE
DOI 10.1109/AITest.2019.00-13

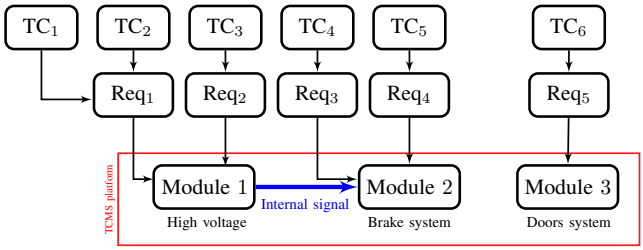

Figure 1: Example of functional dependencies between test cases (TC) based on requirements (Req), modules, and internal signals. Inspired by the TCMS platform.

manual and resource consuming. We believe that automating the gathering of important criteria and utilizing them for providing critical decision support at the integration testing phase has a great potential in reducing testing costs. In addition, previous studies have indicated that ignoring dependencies between integration test cases leads to redundant test executions and unnecessary test failures, increasing the testing efforts [2], [9]. It is also important to note that there is a set of independent test cases in all testing projects. These can be executed at any time in the testing process, meaning that their execution results do not impact the execution of other test cases. Having a general overview of the dependent and independent test cases can thus help test managers schedule the test cases for execution at an early stage of the testing process. In an efficient test schedule, the dependent test cases should be executed at the same time and the independent test cases can be executed at any time during the testing cycle. The cost of re-executing a failed test case is a sum of several factors such as troubleshooting cost, execution cost and installation cost. For instance, the re-execution cost for the failed test cases in Bombardier Transportation (BT) is almost 8 times higher than the execution cost [2]. Therefore, scheduling test cases for execution based on their dependencies reduces the risk of unnecessarily failure between integration test cases. Furthermore, the installation and system setup costs can be decreased if the similar test cases (requiring, e.g., the same initial state) are executed at the same time. The purpose of this paper is to introduce, apply and evaluate a deep learning, NLP-based approach for automatic test case dependency detection. The approach is based on text semantic similarity detection from test case specifications written in natural language. We strive to automatically find dependencies based only on the test specifications, in contrast to finding this information from other sources such as testing records, requirements specification and the internal signals between software modules [2], [3], [10]. Moreover, out of several possible dependencies that can exist, we limit our investigation to functional dependencies. This paper makes the following contributions: (i) clustering dependent test cases into dependency-disjunctive clusters, as well as identifying independent test cases, (ii) the approach is capable of analyzing manual integration test cases written in natural language for a wide range of systems where test case dependencies are a common occurrence, and (iii) evaluation of the approach for a large project in the railway domain at BT using two different clustering algorithms (a hierarchical and a soft clustering), against an independent baseline of precomputed test case dependencies.

II. Background

The dependency between test cases can be categorized into functional dependency, temporal dependency, abstract dependency and causal dependency [11]. In the present work, we aim to automatically detect the functional dependency between manual integration test cases from their natural language test specifications. Let us define the functional dependency between two test cases, within our context, adapted from [11]:

Definition 1. Test cases TC_1 and TC_2 are functionally dependent if they are designed to test different parts of function F_1 or if they are testing the interaction between functions F_1 and F_2.

For instance, given two functions F_1 and F_2 of the same system, let the function F_2 be allowed to execute if its required conditions are already enabled by function F_1. Thus, function F_2 is dependent on function F_1. Consequently, all test cases which are designed to test F_2 should be executed any time after the assigned test cases for testing F_1. Detecting functional dependencies between test cases can lead to a more efficient use of testing resources by means of [2]:

- avoiding redundant execution,
- parallel execution for independent test cases,
- simultaneous execution of test cases that test the same functionality, or
- any combination of the previous options.

However, if the direction of the dependencies between test cases is not detectable, test cases can be categorized into independent and dependent test cases. Splitting all test cases into two main classes can also lead to a more efficient test scheduling in such a way that independent test cases can be executed at any time and also parallel with each other and dependent test cases need be executed at the same time. Previously, we proposed an approach for detecting the functional dependencies between test cases by analyzing the internal signal communications between software modules at BT [10] using the requirement specifications and test case specifications. Fig. 1 provides an overview of the scope of the functional dependency detection by illustrating a part of the train control and management system (TCMS) platform at BT. Fig. 1 shows three software modules from three different sub-systems' functional groups: *high voltage*, *brake system*, and *doors system*. As one can see in Fig. 1, module 1 and module 2 communicate with each other through an internal signal, while they are not communicating with module 3. Thus, module 2 is functionally dependent on module 1 and should be tested after it, and module 3 is independent of both other modules in this example. Thereby, all assigned requirements and test cases for module 2 are functionally dependent on the assigned requirements and test cases for module 1. Fig. 1 also shows the relationships between the assigned requirements and designed test cases to the software modules. Based on Definition 1, we can see that TC_1 and

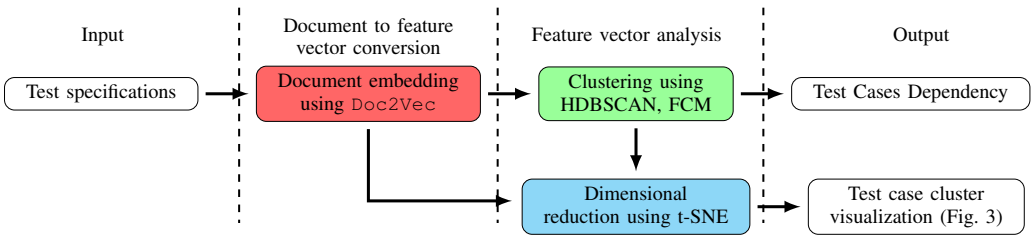

Figure 2: The steps of the proposed approach.

TC_2 are functionally dependent due to shared requirement Req_1 as it implies that they test different aspects of the same function. They are also dependent on TC_3, TC_4, and TC_5 by means of testing the interaction between different functions realized as modules 1 and 2. Since there is no internal signal communication to module 3, TC_6 is an example of an independent test case that tests only one function. Later in this paper, the presented concept of functional dependency between test cases, illustrated in Fig. 1, is utilized as a ground truth (a baseline for comparison) to evaluate the feasibility of the proposed approach in this paper. We need to consider that Fig. 1 represents a part of a bigger figure presented in [10]. However, the utilized inputs (signal information and the requirement specifications) for deriving Fig. 1 are not always available in all testing environments or sometimes the relationships between the software modules, requirements and the test cases are missing due to lack of traceability. Only the manually written test cases are consistently available. Thus, it is beneficial to detect functionally dependent test cases by analyzing the available test case specifications.

III. Related Work

Bates and Horwitz [12] identify components in a modified program testable using old test suite thereby avoiding unproductive testing of unaffected components. The identification of the components is done through program dependence graphs and slicing with test adequacy data. A similar approach to identify changed def.-use pairs for regression testing is taken by Rothermel and Harrold [13]. Our approach presented in this paper is based on different input artifacts. Ryser and Glinz [14] show the importance of managing dependencies and interrelations between scenarios for thorough system testing. They derive test cases by traversing paths in the dependency chart, taking into account data and resource annotations and other specified conditions. A form of directed graph for functionally dependent test cases is proposed by Haidry and Miller [11]. According to this approach, the functionally dependent test cases can be prioritized based on different forms of the graph coverage values. The independent test cases, however, can be executed randomly. The authors remarked that other critical criteria related to test cases should also be considered for test prioritization. The logical dependency between structured requirements is exploited by Arlt et al. [9] to automatically detect redundant test cases. The main goal of their approach is reducing test suites, based on the results of executed test cases, in such a way that a dependent test case gets failed if a corresponding independent test case fails. Acharya et al. [15] proposed a greedy approach for prioritizing test cases in component-based software development environment. From an object interaction graph, which is generated from the UML sequence diagrams for interdependent components, the value of an objective function is evaluated when selecting the test cases for execution. Applying the NLP, machine learning, and big data techniques in the software testing domain is gaining momentum. Specification mining and requirement engineering can be considered as two main applications of NLP in software testing [16]. Research papers exist on the application of text similarity techniques for test selection [17], test prioritization [18] and prioritization of test automation [19]. Furthermore, Ahsan et al. [20] have summarized papers using NLP techniques to generate automated test cases. Sharifi and Hemmati [16] have proposed a NLP-based approach for failure prediction of manual system-level test cases, which can be utilized for test case selection, prioritization and also test suite reduction. `Doc2vec`, proposed by Le and Mikolov [21] is an extension of `Word2Vec` [22] for embeddings from words to word sequences. Empirical evaluations of the quality of document embeddings learnt by `Doc2vec` has shown its robustness [23] and has shown promising results in applications such as intrusion detection [24], e-commerce [25] and news classification [26].

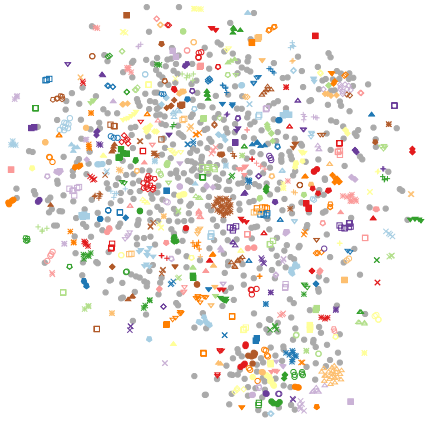

Figure 3: Clustered test cases using the HDBSCAN algorithm, each color-shape represent a cluster of test cases, the gray circles represent independent test cases.

IV. The proposed Approach

From observing the test case specifications provided by BT, we have noticed that functionally dependent test cases often have similar structure, sometimes common setup and teardown routines, with the relevant modules being adjusted for the specific test cases. This meant that within the context of test case specifications, references to functionally dependent elements are likely to occur in similar contexts. We call such test cases as being *semantically similar*. Thus, in our context, semantic similarity between test case specifications correlates with functional dependency between the test cases. This section presents our novel approach for functional dependency detection from natural language test case specifications. Fig. 2 shows the stages of the proposed approach. In the first stage, we derive the feature vectors[1] from the test specifications. The main characteristic of these feature vectors is that semantically similar documents have similar vectors according to some similarity function. In the next stage, the vectors are clustered using that similarity function, forming clusters of feature vectors corresponding to clusters of test cases. Even though this approach does not detect the direction of functional dependencies, it makes it feasible to manually reconstruct them by reducing the problem scope to elements of clusters instead of the entire set.

A. Feature Vector Generation

We have applied the `Doc2Vec`[2] algorithm to generate document embeddings of the test specifications. Document (word) embedding generally represents a set of language modeling techniques in the area of NLP. The embedding models map words, sentences, paragraphs or phrases to vectors of real numbers [28]. The `Doc2Vec` and the other similar methods employ a single layer neural network (neural embeddings) [27] to construct embeddings. Basically, `Doc2Vec` is trained to reconstruct linguistic contexts of words for each document. It takes, as its input, a large corpus of text, in our context test case specifications written in English, and produces a vector space, typically of several hundred dimensions, with each unique word and document in the corpus being assigned a corresponding vector in the space [29]. One way to utilize the document embeddings of `Doc2Vec` is to compute the cosine similarity[3] between them which corresponds to the semantic similarity of the documents in question. For instance, in the provided example in Fig. 1, the test cases are designed to test the interaction between the *high voltage* and the *brake system*. Thus, a set of common keywords and similar features are repeated in the texts of TC_1 to TC_5, which describes the functionality of *high voltage* and the *brake system*. The `Doc2Vec` algorithm computes the feature vector for TC_1, TC_2, TC_3, TC_4, TC_5 and TC_6, and those test cases which are more similar to each other, have a greater cosine similarity between their vectors.

[1]A feature vector is required for a numerical representation of symbolic features in the test specification document.

[2]`Doc2Vec`, also known as `Paragraph Vectors`, is an unsupervised algorithm that generates feature vectors for sentences, paragraphs or documents [27].

[3]Cosine similarity between two vectors is a measure which calculates the cosine of the angle between the vectors [27].

B. Clustering Feature Vectors

Clustering is a method for classifying a number of objects which have several attributes within different classes [30]. The main goal of clustering is sorting the most similar objects into the same class and therefore the members of different clusters are dissimilar. Clustering high dimensional data points is still a significant challenge in the field of data mining [31]. As mentioned before, after running the `Doc2Vec` algorithm, a set of high-dimensional vectors (representing feature vectors of each test case) is generated. Our goal is to categorize together similar feature vectors (test cases), where similarity is expressed as higher cosine similarity. Since each vector is representing a test case, a set of clusters containing test cases can be proposed for execution. All test cases are not necessarily dependent or similar to each other, meaning that a set of independent test cases can exist in testing projects. As outlined in Section II, it is equally important to ensure that independent test cases are identified. On the other hand, the dependent test cases can be divided into several dependency groups, for instance test case TC_4 in Fig. 1 belongs to two clusters, cluster 1 which contains the designed test cases for testing module 1 and also cluster 2 that contains the designed test cases for testing module 2. This kind of classification is called soft clustering, where each data point can potentially belong to multiple clusters [32]. In essence, we need a fast and robust clustering algorithm, which can handle the high dimensionality of data in our context. The clustering algorithm should be able to propose a set of non-clusterable data points (independent test cases) and also data points for soft clustering. In this paper, we apply and evaluate two clustering algorithms: HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise) and FCM (Fuzzy c-means). The essential reason for utilizing HDBSCAN is its ability to dealing with the high dimensional data. However, the FCM considers each object a member of every cluster, with a variable degree of membership [33], which make sense in the testing domain. In other word, each test case can be dependent to more than one test case. Later in this paper, the performance of clustering algorithms is evaluated by comparing with the ground truth. The HDBSCAN algorithm measures the distance between the vectors and provides automatically number of clusters, the cluster themselves, and also a set of non-clusterable vectors (noise) [34]. Other clustering algorithms are designed to cluster every vector to some clusters, which indicates their inability to handle outliers, corrupt data and noise in the clustering process. The non-clusterable vectors provided by the HDBSCAN can be interpreted as independent test cases, to be executed in no particular order. This hypothesis is analyzed and discussed further in Section VI. In addition, each cluster consists of a set of test cases which are dependent and must be tested together. The FCM algorithm provides a set of clusters where each data point can belong to multiple clusters. The FCM assigns a membership grade to each cluster for each data point, where each membership grade represents the extent to which a data point belongs to a specific cluster [33]. In the upcoming sections, both HDBSCAN and FCM are applied on

an industrial testing project at BT and their performance is compared with the ground truth.

V. Empirical Evaluation

This section presents an empirical evaluation of the proposed approach. The following research question is addressed in the empirical evaluation:

RQ. *What is the performance of the proposed test dependency detection approach in comparison with the ground truth when clustered using HDBSCAN and FCM?*

A. Industrial Case Study

The feasibility of the proposed approach has been evaluated by studying an ongoing testing project for the underground subway train in Stockholm, called BR490[4] project, at BT [35]. The units of analysis in this case study are test cases (test specifications) at the integration testing level for the BR490 project. A total of 1748 test cases, which are designed for testing different functional groups (e.g. *brake system*, *air conditioning*, *doors system*, *battery power supply*), have been extracted from the DOORS (Dynamic Object-Oriented Requirements System) database at BT. The extracted test cases are converted into vectors using paragraph-vectors implementation [36] of the `Doc2Vec` model, and then the test cases are clustered by applying the HDBSCAN and FCM [34] algorithms. As highlighted earlier, the `Doc2Vec` algorithm generates a set of data points in the n-dimensional space. Using t-SNE (t-distributed stochastic neighbor embedding [37]) for dimensionality reduction and visualization helped us to create an overview of the density range of the generated data points. Fig. 3 shows the results of clustering using the HDBSCAN and FCM algorithms, respectively. HDBSCAN automatically determined that there are 393 clusters (with different cardinality) for dependent test cases, illustrated in Fig. 3. Moreover, 526 data points are detected as non-clusterable data points (the gray circles in Fig. 3). Since there is no known optimal number of clusters when applying unsupervised learning, the problem formulation can specify the number of clusters. The FCM clustering algorithm requires determining the number of clusters in advance. Note that our final goal is comparing the performance accuracy of the proposed clustering algorithms, therefore we have set the FCM to group all data points into 393 clusters. This resulted in 171 clusters with more than zero elements and 222 empty clusters, that have been removed from the samples. However, the FCM does not report any data points as non-clusterable, noise or outliers. Since executing FCM results in a set of probabilities for each point belonging to each cluster, we assigned each point to the cluster with the highest probability. In practice, we have used the default behavior of the algorithm as much as possible. The main decision was to set the number of dimensions for feature vector generation to 128 in order to improve the performance of the approach. The rest of the parameters used were 100 training epochs, batch size of 32, 2 noise words to sample from the noise distribution and learning rate of 0.001. Since our implementations of HDBSCAN and FCM use Euclidean distances instead of cosine similarity, we have approximated the cosine similarity by applying L2-normalization on the entire vector space. HDBSCAN was used with minimum cluster size of 2 and FCM was used with array exponentiation coefficient of 2, stopping error criterion of 0.005 and maximum of 1000 iterations. The source code of our work can be found online at [38], together with anonymized feature vectors and a test-case requirement graph. As the result of this approach we have determined clusters of similar test cases (Fig. 3). Within the clusters, the similarity can be based on any combination of the similarity between preconditions, test steps, or postconditions.

[4]The BR490 series is an electric rail car specifically for the S-Bahn Hamburg GmbH network in production at Bombardier Hennigsdorf facility.

B. Ground Truth

As explained earlier, the dependency between test cases has been detected by applying different methods and using several testing artifacts such as internal signals [10], requirement specifications, test records [2] and also questionnaire-based studies [3]In [10], we analyzed the BR490 project's internal signals to the implemented modules (see Fig. 1) for detecting dependencies between requirements and thereby identifying the functional dependencies between test cases. By using this information, we have derived a graph for dependencies between requirements and test cases in the BR490 project at BT. The empirical evidence in [10] is used in this work as the ground truth. In the derived dependency graph each test case depends on zero or more requirements. Each of the requirements, in turn, can depend on zero or more other requirements. Moreover, multiple test cases can depend on the same requirement. A total of 3938 requirements and 1748 test cases from 17 different sub-level function groups are present in our ground truth. Since our method produces clusters of test cases, it was necessary to find a unified measure for comparison of the dependency graph and the set of clusters. We achieved this by considering all test cases and requirements as nodes in a graph, with the edges of this graph being dependencies between the test cases and requirements.

(a) Distance 2 (b) Distance 4

Figure 4: Examples of test case to test case distances. Red circles are test cases, and blue squares are requirements.

We computed the shortest path between any two test cases that passes through other test cases or requirements (Fig. 4). Thus, if the test cases TC_A and TC_B have a distance of 2, then they share a common requirement, as illustrated in Fig. 4a, and Fig. 5. The distance of two is the most significant for the industrial applications since it clearly defines the test cases that commonly operate on the same interfaces and thus are suitable for optimization of the testing schedule. Fig. 5 is extracted from Fig. 3 as an example of two test cases which are located in the same cluster. As we can see, Fig. 5 represents a part of two test specifications (`exterior access safe 006` and `exterior access safe 001`) which both test different parts of the same function, as shown by the presence

	HDBSCAN				FCM				Random				Ground Truth	
D	TP	TN	FP	FN	TP	TN	FP	FN	TP	TN	FP	FN	Indep.	Dep.
2	390	1524482	1758	248	247	1444805	81435	391	338	763456	762784	300	1526240	638
3	393	1523710	1755	1020	358	1444141	81324	1055	712	763055	762410	701	1525465	1413
4	493	1520076	1655	4654	1095	1441144	80587	4052	2562	761171	760560	2585	1521731	5147
5	498	1515925	1650	8805	1780	1437673	79902	7523	4627	759080	758495	4676	1517575	9303
6	522	1506980	1626	17750	3238	1430162	78444	15034	9124	754608	753998	9148	1508606	18272
7	532	1499037	1616	25693	4385	1423356	77297	21840	13038	750569	750084	13187	1500653	26225
8	538	1486471	1610	38259	6163	1412562	75519	32634	19378	744337	743744	19419	1488081	38797
9	549	1475985	1599	48745	7346	1403248	74336	41948	24565	739027	738557	24729	1477584	49294
10	562	1462104	1586	62626	8675	1390683	73007	54513	31486	732054	731636	31702	1463690	63188

Table I: Comparison of detected dependencies from HDBSCAN clusters, FCM clusters, and random assignments with the ground truth based on 1526878 test case pairs. TP: True Positive, TN: True Negative, FP: False Positive and FN: False Negative.

of the same requirement (`sr-C30-srs-safe-req-226`). Note that even though the requirements are present in the ground truth, they were removed before application of our method to simulate the conditions where such information is completely unavailable.

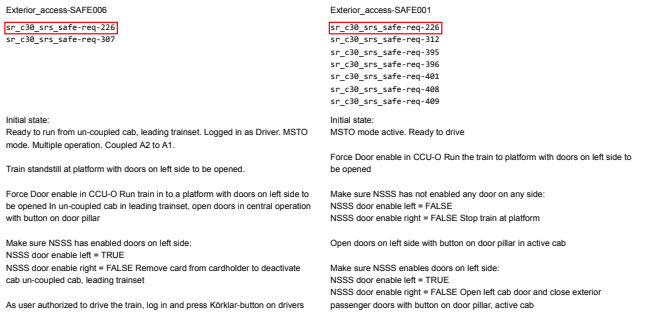

Figure 5: Example of two test cases with a common requirement.

VI. Results

We compared the independency and dependency results between test cases grouped into clusters in Fig. 3 with the ground truth results.

A. Comparing the Clustering Results with the Ground Truth

In order to compare the results achieved by the clustering algorithms with the ground truth, we established a pairwise comparison between each two test cases. Given the total number of 1748 test cases, there are $\binom{1748}{2} = 1\,526\,878$ test case pairs. For each test case pair, we calculated the distance between the test cases within the ground truth dependency graph. By varying the maximum distance (D) for which we consider two test cases to be interdependent, we generated a number of interpretations of the ground truth. The number of dependent (True) and independent (False) test case pairs for each distance D from 2 to 10 is shown in Table I. If the maximum distance is set to 1 or lower, all test cases become fully independent, while the maximum found distance is 22. For distances from 2 to 10, Table I compares dependency detection quality of the HDBSCAN and FCM clusters. Moreover, the performance measurement results of the proposed clustering algorithms (HDBSCAN and FCM) are compared with three baseline approaches (i) a random classifier, (ii) a classifier that always claims dependency (All-Dep), and (iii) a classifier that always claims that there is no dependency (No-Dep). The results follow similar trends for $D > 10$ and remain unchanged after $D = 22$ due to lack of larger distances in the ground truth dependency graph. The results for $11 \leq D \leq 22$ have been omitted from the presented results. The distance 2 means that, for the ground truth, if two test cases are to be considered dependent, they have to be dependent on the same requirement (as illustrated in Fig. 4a). If the shortest path between two test cases is 3 or more, the test cases are considered independent of each other.

B. Performance Metric Selection

Choosing suitable performance metrics is critical and also influences the measured performance of our approach. The performance metric used is dependent on the intended use as well as the balance between positive and negative instances. A pairwise approach, as is used in this paper, exacerbates the already relative sparseness of dependent test cases and thus the imbalance of the data set. It is well known that using accuracy as the performance metric for imbalanced data sets can result in misleading results [39]. Therefore, we have opted to primarily measure the performance of our proposed approach using Precision, Recall, F1 Score, and Cohen's Kappa [40]. The mentioned performance metrics are selected based on the following considerations: (i) identifying independent test cases (true negative) is as important as identifying dependent ones, (ii) the number of missing dependencies between test cases outweighs the number of dependencies. Precision and Recall are commonly used metrics to measure the accuracy of a binary prediction system, F1 Score is a measure of a test's accuracy, which is defined as the harmonic mean of Precision and Recall and does not directly include true negatives (independent test cases). Cohen's Kappa is a measure of agreement between two observers and it is less than or equal to 1. Values of 0 or less imply a useless classifier [41]. Table II shows the computed results of the mentioned performance metrics, where the best results for each metric are highlighted.

C. Metric Comparison

In this subsection we examine each of the metrics shown in Table II individually.

a) Precision: represents the number of correctly detected dependencies over the total number of detected dependencies by the proposed approach. The HDBSCAN has continuously the highest value followed by the FCM method. 0.1816 of the identified dependencies by the HDBSCAN are detected

	Precision				Recall			F1 Score				Cohen's Kappa		
D	HDB	FCM	RND	All-Dep	HDB	FCM	RND	HDB	FCM	RND	All-Dep	HDB	FCM	RND
2	**0.1816**	0.0030	0.0004	0.0004	**0.6113**	0.3871	0.5298	**0.2800**	0.0060	0.0009	0.0008	**0.2795**	0.0052	0.0001
3	**0.1830**	0.0044	0.0009	0.0009	0.2781	0.2534	**0.5039**	**0.2207**	0.0086	0.0019	0.0018	**0.2199**	0.0068	0.0000
4	**0.2295**	0.0134	0.0034	0.0034	0.0958	0.2127	**0.4978**	**0.1352**	0.0252	0.0067	0.0067	**0.1334**	0.0190	0.0000
5	**0.2318**	0.0218	0.0061	0.0061	0.0535	0.1913	**0.4974**	**0.0870**	0.0391	0.0120	0.0121	**0.0849**	0.0285	−0.0001
6	**0.2430**	0.0396	0.0120	0.0120	0.0286	0.1772	**0.4993**	0.0511	**0.0648**	0.0234	0.0237	**0.0487**	0.0461	0.0000
7	**0.2477**	0.0537	0.0171	0.0172	0.0203	0.1672	**0.4972**	0.0375	**0.0813**	0.0330	0.0338	**0.0350**	0.0567	−0.0002
8	**0.2505**	0.0755	0.0254	0.0254	0.0139	0.1589	**0.4995**	0.0263	**0.1023**	0.0483	0.0496	0.0237	**0.0703**	0.0000
9	**0.2556**	0.0899	0.0322	0.0323	0.0111	0.1490	**0.4983**	0.0213	**0.1122**	0.0605	0.0625	0.0187	**0.0749**	−0.0002
10	**0.2616**	0.1062	0.0413	0.0414	0.0089	0.1373	**0.4983**	0.0172	**0.1198**	0.0762	0.0795	0.0145	**0.0767**	−0.0002

Table II: Precision, Recall, F1 Score, and Cohen's Kappa metrics based on the data from Table I. The best results are highlighted. All values are between -1 and 1, HDB and RND are abbreviation for HDBSCAN and Random classifier.

	HDBSCAN				FCM			
D	Precision	Recall	Accuracy	F1 Score	Precision	Recall	Accuracy	F1 Score
2	**0.9981**	**0.6113**	**0.8050**	**0.7582**	0.8789	0.3871	0.6668	0.5375
3	**0.9959**	**0.2781**	**0.6385**	**0.4348**	0.8275	0.2534	0.6002	0.3879
4	**0.9891**	0.0958	0.5474	0.1747	0.8021	**0.2127**	**0.5801**	**0.3363**
5	**0.9804**	0.0535	0.5262	0.1015	0.7845	**0.1913**	**0.5694**	**0.3076**
6	**0.9628**	0.0286	0.5137	0.0555	0.7727	**0.1772**	**0.5625**	**0.2883**
7	**0.9476**	0.0203	0.5096	0.0397	0.7653	**0.1672**	**0.5580**	**0.2745**
8	**0.9287**	0.0139	0.5064	0.0273	0.7583	**0.1589**	**0.5541**	**0.2627**
9	**0.9104**	0.0111	0.5050	0.0220	0.7477	**0.1490**	**0.5494**	**0.2485**
10	**0.8913**	0.0089	0.5039	0.0176	0.7340	**0.1373**	**0.5438**	**0.2313**

Table III: The averages across 100 observations of performance metrics after random undersampling to a $50:50$ ratio.

correctly. For No-Dep method, Precision always results in 0 and thus has been omitted from Table II.

b) Recall: is the number of correctly detected dependencies over the total number of existing dependencies. For the All-Dep approach Recall is always equal to 1 and for the No-Dep approach it is always 0 and therefore have been omitted from Table II. For $D = 2$, the HDBSCAN outperforms all other shown approaches and this correlates with the previous conclusion that the distance $D = 2$ is the most industrially applicable interpretation of the ground truth.

c) F1 Score: is a harmonic average of Precision and Recall and is used as compound measurement. Its best value is 1 equating to the perfect Precision and Recall. According to Table II the highest F1 Score value is archived for $D = 2$ using the HDBSCAN algorithm, however, our alternative clustering approach, FCM, outperforms it at $D = 6$. Moreover, both proposed clustering algorithms reach a better value for F1 Score, compared to the random, All-Dep, and No-Dep classifier.

d) Cohen's Kappa: can be considered as a robust measurement of classifiers with imbalanced data sets [42]. As we can see in Table II using the HDBSCAN algorithm for clustering leads to the largest value of Cohen's Kappa, 0.2795, compared to FCM and random approach [41]. We need to consider that All-Dep and No-Dep approaches result in Kappa value of 0 and therefore they have been omitted from Table II.

Comparing the performance measurement results of the proposed clustering algorithms with each other and with the ground truth shows that the HDBSCAN generally provides better results for clustering in terms of functional dependencies between test cases. The primary reason for this result is HDBSCAN's ability for identifying non-clusterable data points. Even though the classification methods are only fairly good, they are still sufficient to make improvements in industrial settings, by identifying the functional dependencies.

D. Random Undersampling strategy for imbalanced datasets

The ratio of dependent and independent test case pairs in the ground truth is highly biased towards the independent pairs. It varies depending on the observed maximum dependency depth, from 99.96% for distance 2 to 92.79% for distance 22. This imbalance does not affect the training of our system since the ground truth is not used for the training.

However, two popular evaluation metrics F1 Score and Accuracy are tailored towards balanced data [42] and thus we present these metrics for re-balanced data set. We randomly undersampled the majority class (independent test case pairs) to a $50:50$ post sampling class distribution. To avoid random sampling bias, we have sampled each group 100 times and computed averages and standard deviations. Table III shows that the average of F1 Score metric across 100 observations is 0.7582 and 0.5375 for the HDBSCAN and FCM algorithms respectively at $D = 2$, which is a significant improvement compared with the results before random undersampling. Standard deviations are less than 0.02.

VII. Conclusion

In this paper, we proposed a deep-learning based natural language processing approach to analyze the correlation between test case text semantic similarities and their functional dependencies. The test cases are interpreted as a set of vectors in n- dimensional space using `Doc2Vec` algorithm. Two clustering algorithms (HDBSCAN and FCM) were applied on the generated vectors in order to classify the dependent test cases into clusters. The proposed approach has been applied to an industrial use case at Bombardier Transportation AB. The performance of the proposed clustering algorithms (HDBSCAN and FCM) is compared with a random classifier, a classifier that always claims dependency and also a classifier that always claims that there is no dependency. Moreover, Precision, Recall,

F1 Score, and Cohen's Kappa are selected as the performance metrics. In order to deal with imbalanced data sets distribution, random undersampling is applied on the results. We have re-measured Precision, Recall, and F1 Score metrics together with the Accuracy metric on the randomly undersampled data sets. The results indicate that, using the HDBSCAN algorithm obtains the accuracy level around 80% and the F1 Score of up to 75%. The results of the empirical evaluation show that the proposed approach has a more accurate performance with the HDBSCAN clustering algorithm.

REFERENCES

[1] W. Afzal, S. Alone, K. Glocksien, and R. Torkar, "Software test process improvement approaches: A systematic literature review and an industrial case study," *Journal of Systems and Software*, vol. 111, pp. 1 – 33, 2016.

[2] 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.

[3] 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.

[4] S. Tahvili, M. Saadatmand, and M. Bohlin, "Multi-criteria test case prioritization using fuzzy analytic hierarchy process," in *The 10th Int. Conf. on Software Engineering Advances*, 2015.

[5] P. Resnik, "Using information content to evaluate semantic similarity in a taxonomy," in *In Proceedings of the 14th Int. Joint Conf. on Artificial Intelligence*, 1995.

[6] J. Kwon, C. Moon, S. Park, and D. Baik, "Measuring semantic similarity based on weighting attributes of edge counting," in *Artificial Intelligence and Simulation*, T. G. Kim, Ed., 2005.

[7] Y. Cai, Q. Zhang, W. Lu, and X. Che, "A hybrid approach for measuring semantic similarity based on ic-weighted path distance in wordnet," *Journal of Intelligent Information Systems*, vol. 51, no. 1, pp. 23–47, Aug 2018.

[8] M. Hadj Taieb, M. Ben Aouicha, and Y. Bourouis, "Fm3s: Features-based measure of sentences semantic similarity," in *Hybrid Artificial Intelligent Systems*, 2015.

[9] 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 Int. Conf. on Software Testing, Verification and Validation*, 2015.

[10] 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.

[11] S. Haidry and T. Miller, "Using dependency structures for prioritization of functional test suites," *IEEE Transactions on Software Engineering*, vol. 39, no. 2, pp. 258–275, 2013.

[12] S. Bates and S. Horwitz, "Incremental program testing using program dependence graphs," in *Proceedings of the 20th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 1993.

[13] G. Rothermel and M. Harrold, "Selecting tests and identifying test coverage requirements for modified software," in *Proceedings of the Int. Symposium on Software Testing and Analysis*, 1994.

[14] J. Ryser and M. Glinz, "Using dependency charts to improve scenario-based testing - Management of inter-scenario relationships: Depicting and managing dependencies between scenarios," in *Proceedings of the 17th Int. Conf. on Testing Computer Software*, 2000.

[15] A. Acharya, D. P. Mohapatra, and N. Panda, "Model based test case prioritization for testing component dependency in cbsd using uml sequence diagram," *Int. Journal of Advanced Computer Science and Applications*, vol. 1, no. 3, pp. 108–113, 2010.

[16] H. Hemmati and F. Sharifi, "Investigating nlp-based approaches for predicting manual test case failure," in *2018 IEEE 11th Int. Conf. on Software Testing, Verification and Validation*, April 2018, pp. 309–319.

[17] M. Unterkalmsteiner, T. Gorschek, R. Feldt, and N. Lavesson, "Large-scale information retrieval in software engineering - an experience report from industrial application," *Empirical Software Engineering*, vol. 21, no. 6, pp. 2324–2365, 2016.

[18] S. Thomas, H. Hemmati, A. Hassan, and D. Blostein, "Static test case prioritization using topic models," *Empirical Software Engineering*, vol. 19, no. 1, pp. 182–212, 2014.

[19] D. Flemström, P. Potena, D. Sundmark, W. Afzal, and M. Bohlin, "Similarity-based prioritization of test case automation," *Software Quality Journal*, pp. 1–29, 2018.

[20] I. Ahsan, W. Butt, M. Ahmed, and M. Anwar, "A comprehensive investigation of natural language processing techniques and tools to generate automated test cases," in *Proceedings of the Second Int. Conf. on Internet of Things, Data and Cloud Computing*, 2017.

[21] Q. Le and t. Mikolov, "Distributed representations of sentences and documents," in *Proc. of the 31st Int. Conf. on Machine Learning*, 2014.

[22] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proceedings of the 26th Int. Conf. on Neural Information Processing Systems*, 2013.

[23] J. Lau and T. Baldwin, "An empirical evaluation of doc2vec with practical insights into document embedding generation," in *Proceedings of the 1st Workshop on Representation Learning for NLP*. Association for Computational Linguistics, 2016.

[24] M. Mimura and H. Tanaka, "Reading network packets as a natural language for intrusion detection," in *Info. Security and Cryptology*, 2018.

[25] V. Phi, L. Chen, and Y. Hirate, "Distributed representation-based recommender systems in e-commerce," in *Proceedings of the 8th forum on data engineering and information management*, 2016.

[26] L. Trieu, H. Tran, and M. Tran, "News classification from social media using twitter-based doc2vec model and automatic query expansion," in *Proceedings of the Eighth Int. Symposium on Information and Communication Technology*, 2017.

[27] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *CoRR*, vol. abs/1301.3781, 2013.

[28] M. Pelevina, N. Arefyev, C. Biemann, and A. Panchenko, "Making sense of word embeddings," *CoRR*, vol. abs/1708.03390, 2017.

[29] O. Levy and Y. Goldberg, "Neural word embedding as implicit matrix factorization," in *Proceedings of the 27th Int. Conf. on Neural Information Processing Systems - Volume 2*, 2014.

[30] R. Namayandeh, F. Didehvar, and Z. Shojaei, "Clustering validity based on the most similarity," *CoRR*, vol. abs/1302.3956, 2013.

[31] M. Steinbach, L. Ertöz, and V. Kumar, *The Challenges of Clustering High Dimensional Data*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 273–309.

[32] M. Ahmed, S. Yamany, N. Mohamed, A. Farag, and T. Moriarty, "A modified fuzzy c-means algorithm for bias field estimation and segmentation of mri data," *IEEE Transactions on Medical Imaging*, vol. 21, no. 3, pp. 193–199, 2002.

[33] J. Bezdek, R. Ehrlich, and W. Full, "Fcm: The fuzzy c-means clustering algorithm," *Comput Geosci*, vol. 10, no. 2, pp. 191–203, 1984.

[34] L. McInnes, J. Healy, and S. Astels, "HDBSCAN: Hierarchical density based clustering," *Journal of Open Source Software*, vol. 2, no. 11, 2017.

[35] Electric Multiple Unit Class 490 – Hamburg, Germany, url = https://www.bombardier.com/en/transportation/projects/project.ET-490-Hamburg-Germany.html. [Accessed: 2018-02-13].

[36] N. Ilenic, "A pytorch implementation of paragraph vectors (doc2vec)," https://github.com/inejc/paragraph-vectors, 2017.

[37] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008.

[38] L. Hatvani and S. Tahvili, "Clustering dependency detection," https://github.com/leohatvani/clustering-dependency-detection, 2018.

[39] H. Han, W.-Y. Wang, and B.-H. Mao, "Borderline-smote: A new over-sampling method in imbalanced data sets learning," in *Advances in Intelligent Computing*, 2005.

[40] D. Powers, "Evaluation: From precision, recall and f-measure to roc., informedness, markedness & correlation," *Journal of Machine Learning Technologies*, vol. 2, no. 1, pp. 37–63, 2011.

[41] A. J. Viera and J. M. Garrett, "Understanding interobserver agreement: the kappa statistic." *Family medicine*, vol. 37 5, pp. 360–3, 2005.

[42] L. A. Jeni, J. F. Cohn, and F. De La Torre, "Facing imbalanced data–recommendations for the use of performance metrics," in *Proceedings of the 2013 Humaine Association Conf. on Affective Computing and Intelligent Interaction*. IEEE Computer Society, 2013.