Mälardalen University
School of Innovation Design and Engineering
Västerås, Sweden

Thesis for the Degree of Bachelor in Computer Science 15.0 credits

EVALUATION OF THE CORRELATION BETWEEN TEST CASES DEPENDENCY AND THEIR SEMANTIC TEXT SIMILARITY

Filip Andersson
fan15003@student.mdh.se

Examiner: Wasif Afzal
Mälardalen University, Västerås, Sweden

Supervisors: Sahar Tahvili
Mälardalen University, Västerås, Sweden

Leo Hatvani
Mälardalen University, Västerås, Sweden

May 19, 2020

Abstract

An important step in developing software is to test the system thoroughly. Testing software requires a generation of test cases that can reach large numbers and is important to be performed in the correct order. Certain information is critical to know to schedule the test cases incorrectly order and isn't always available. This leads to a lot of required manual work and valuable resources to get correct. By instead analyzing their test specification it could be possible to detect the functional dependencies between test cases. This study presents a natural language processing (NLP) based approach and performs cluster analysis on a set of test cases to evaluate the correlation between test case dependencies and their semantic similarities. After an initial feature selection, the test cases' similarities are calculated through the Cosine distance function. The result of the similarity calculation is then clustered using the HDBSCAN clustering algorithm. The clusters would represent test cases' relations where test cases with close similarities are put in the same cluster as they were expected to share dependencies. The clusters are then validated with a Ground Truth containing the correct dependencies. The result is an F-Score of 0.7741*. The approach in this study is used on an industrial testing project at Bombardier Transportation in Sweden.*

Keywords: *Software Testing, Test optimization, NLP, Dependency, Semantic Similarity, Clustering, Cosine Similarity, HDBSCAN.*

Sammanfattning

Ett viktigt steg vid utveckling av mjukvara är att testa systemet noggrant. Att testa mjukvara kräver genererade testfall som kan nå stora antal och är viktiga att utföra i en korrekt ordning. Viss information är kritiskt att känna till för att skedulera testfallen i korrekt ordning och finns inte alltid tillgänglig. Genom att istället analysera deras testspecifikationer skulle det kunna vara möjligt att upptäcka de funktionella kopplingarna mellan testfall. Denna studie presenterar ett språkteknologiskt (NLP) tillvägagångssätt och genomför en klusteranalys på en mängd testfall för att evaluera kopplingen mellan testfallsberoenden och deras semantiska textliknelse. Efter en inledande informationsfiltrering, beräknas testfallens liknelser genom distansfunktionen Cosine. Resultatet av liknelseberäkningen blir sedan klustrad genom klusteralgoritmen HDBSCAN. Klusterna representerar testfallens relationer där testfall med stor liknelse placerades i samma kluster som de förväntades dela krav med. Klusterna är sedan validerade med de korrekta resultaten som omfattar de korrekta kraven. Resultatet är ett F-Score på 0.7741*. Tillvägagångssättet av denna studie används på ett industriell testprojekt hos Bombardier Transportation.*

Nyckelord: *Mjukvarutestning, Testoptimering, NLP, Beroenden, Semantisk liknelse, Kluster, Cosine, HDBSCAN.*

Acknowledgements

First and foremost, I have to thank the people who helped to make this thesis possible. I would like to thank my supervisors Sahar Tahvili and Leo Hatvani for their guidance throughout the study and their knowledge in the area. I would also like to express my gratitude to Ola Sellin and Bombardier Transportation for the opportunity to perform this study in collaboration with you.

I would also like to thank my family and best friend Franchesca, for their never-ending love, support, and motivation throughout my university years and beyond. Thank you for always being there for me and for teaching me to never stress and worry about things I can't change, but to rather make the best of the situation and prepare for future obstacles.

This work was supported in part by VINNOVA through TESTOMAT projects.

Contents

1 Introduction

In recent years, optimizing the process of software testing and assure high quality of the system has received a lot of attention. Software testing is an expensive and effort-intensive phase and test optimization is seen as a multiple-criteria decision-making problem. In fact, several criteria must be clarified and fulfilled in an early stage of a testing process [1], [2], [3]. Software testing relies on requirements that are written in natural language, and later on manually transformed into test cases that are executed manually by human testers.

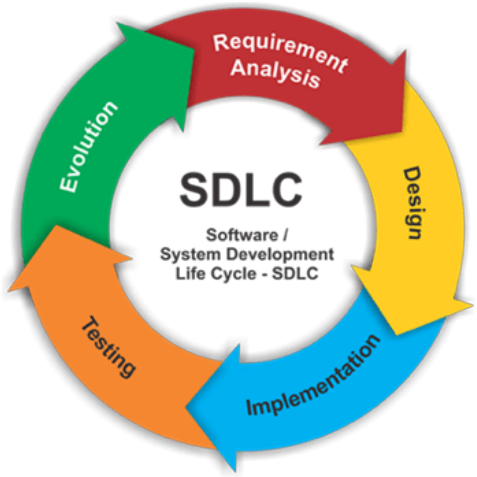


Figure 1: Software Development Life Cycle.

Figure 1 illustrates the main phases of a typical software development life cycle. The cycle starts with gathering and analysing the requirements for the system. The second phase involves designing the system followed by the implementation of it. After that, the system needs to be tested. This is usually done in different levels to detect more errors or flaws in the product, evaluate if the system accomplishes the required needs as well as help identify areas of the system that is yet to be tested. Unit testing, integration testing, system testing, and acceptance testing are the four main levels of system testing. And finally, developing and maintaining the system which will lead to more requirements, and therefore starting the cycle over again [1].

Measuring the effect of the critical criteria on each test case is a challenging task, especially in a manual test procedure where all testing artifacts are written in natural language text by humans. However, the number of critical criteria on the testing process is dependent on multiple factors e.g. testing level (unit, integration, system), testing procedure (manual, semi-automated, fully automated).

Until now, following criteria are identified by the researches which can be utilized for the test optimization purposes: dependency between test cases [4], requirement coverage, execution time [5], [6] and test case's similarities [7], [8]. By knowing the relationship and dependencies between the test cases, it is possible to enhance the prioritization of their execution. Not only can test cases fail if their dependencies are overlooked, but the relationships can be beneficial for multiple test optimization purposes such as test case selection, prioritization, scheduling as well as also test suite minimization. In order to detect the relations between the test cases, a lot of important information is needed about the system under test, the requirement specifications, levels of testing, etc. This information is not always available in all test procedures as a lot of the time the test case specification is the only information available.

In the literature, a lot of Natural Language Processing (NLP) based approaches have been proposed in order to minimize the manual efforts of converting the requirements into test cases. In this thesis, we apply and evaluate an NLP-based approach to find the dependencies between test cases. In other words, we aim to evaluate the correlation between the test cases' dependencies and their semantic text similarities. Test case selection, prioritization, and test scheduling can be considered as an application of this study.

Moreover, the feasibility of the proposed approach in this study is analyzed on an industrial testing project at Bombardier transportation in Sweden.

1.1 Problem Formulation

As highlighted earlier, the main focus of this thesis is evaluating the correlation between test case dependency and their text semantic similarities. Moreover, the intent is to investigate if the prioritization of test case execution can be enhanced by processing their text specification, as well as compare how precise the results are compared to manual work. It is anticipated that the results of this work can be used to optimize software testing procedures in real-life scenarios to save resources like time and money from this demanding process, as well as being used for future studies in related subjects. However, this thesis seeks to address the following research question:

To which extent the functional dependencies between test cases can be detected through analyzing their test specification?

Moreover, to evaluate the performance of the proposed solution, the ground truth (GT) presented in [9] for the functional dependency between test cases is conducted at Bombardier Transportation (BT).

The study is laid out as followed: Section 2 provides a background of the initial problem, test optimization, and natural language processing. Section 3 presents related work and studies that previously have been done in the subject area. Section 4 describes the utilized techniques to complete the study and answer the research question. Section 5 mentions the ethical and societal considerations taken into account in the study. Section 6 presents the study's procedure of applying the chosen NLP techniques for clustering and validating the result. Section 7 presents the result from the resulting clustering as well as the validation when compared to the Ground Truth. Section 8 highlights some threats to validity and discusses the outcome of the study, as well as suggests some future directions of the present work. And lastly, Section 9 concludes this study.

2 Background

This chapter provides a background to the study. Section 2.1 describes the initial problem and the importance of test optimization. Section 2.2 presents a natural language processing-based approach of performing cluster analysis.

2.1 Test case dependencies

An important step in developing software is to test the system thoroughly to determine if everything works as expected and also detecting as many hidden bugs as possible in the system under test. There are several procedures for testing software today: manual testing, semi-automated, and fully automated testing. For testing a software product, we need to generate test cases. However, the number of required test cases for testing a software product can be pretty large, and executing all generated test cases is not an efficient decision. Creating independent test cases has several advantages, but when testing interfaces between different modules, there need to be sets of test cases that test various parts of the combined interfaces. These test cases are functionally dependent on each other and have effects on the execution of each other. Therefore, the test cases must be performed in a correct order, because otherwise there may arise problems with the test execution, e.g. wasting time and testing resources [1], [9].

Certain information is critical to know to schedule the test cases in correct order, for example the system architecture, the dependencies between the different test cases, what requirements each of the test cases cover as well as test case's execution time. This critical information, as well as other information that could be important in relation to testing of the software, is not always available. Instead, simply the requirement specification is the only information available that does not describe this information enough or at all. This leads to a lot of needed manual work and valuable resources in order to get it correct. To analyze the requirement specifications faulty can lead to an ineffective testing process with the same consequences as not caring about a correct scheduling at all, and thereby wasting resources. Changes to the system under test can cause more test cases to occur and thereby new dependencies. With an effective test procedure, it is possible to detect errors at an earlier stage in the development that can be corrected sooner to achieve results of higher quality [10].

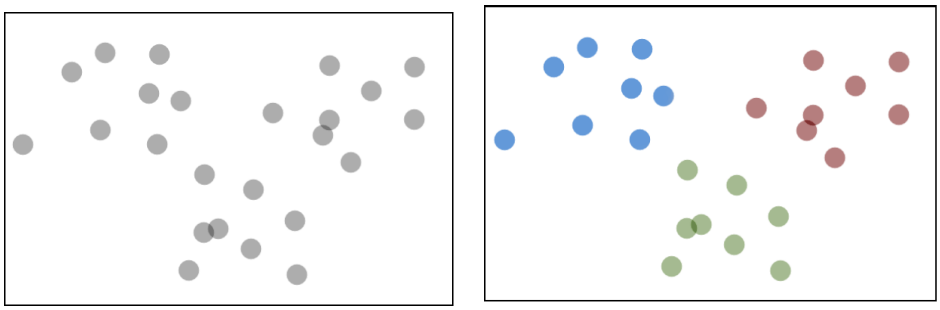
Researchers have identified several kinds of test case dependencies [9]:

- **Functional dependency:** Represents the interactions between the system functionality and their run sequences. If function F1 enabled the required preconditions of function F2, then the function F2 is dependant on the function F1. Therefore, all test cases that test F2 should execute after the test cases assigned with testing F1.
- **Temporal dependency:** Represents the execution sequence between test cases. It occurs mostly in real-time system contexts. If test case TC2 depends on test case TC1, then TC2 should be tested right after TC1.
- **Abstract dependency:** Occurs in the models with a hierarchical decomposition of the model elements. Test cases are hierarchically dependent on each other based on the decomposition structure of the model.
- **Causal dependency:** Can be considered as a different kind of temporal dependency, where specific data items need to be created by TC1 before TC2 is executed.

2.2 Cluster analysis

A cluster analysis is about grouping sets of data so that the data in each cluster are more similar to each other compared to the data in the other clusters. Clustering can have different purposes. An example could be exploratory data analysis, i.e. find interesting regions in data to explore further or get a better idea of what the data looks like. Cluster analysis is under the category of unsupervised machine learning, in other words, the correct answer is unknown - there are no predefined categories that the data should be placed in. A cluster analysis is done through several steps: by performing a feature selection, pre-process the data by measuring the distances between

the data points, cluster the data based on their similarity, and lastly validate the cluster result [11]. Figure 2 shows an example of non-clustered data points compared to clustered data points where Figure 2(a) represents an example of non-clustered and Figure 2(b) shows an example of data points distributed into clusters.



(a) Non-clustered data points (b) Clustered data points

Figure 2: An example of clustering data points.

2.2.1 Feature selection

The first step in performing a cluster analysis is feature selection, where relevant features of a data set are selected for the later steps. This is done partly to remove the features that won't have much relevant impact on the data patterns and thus increase the performance of the algorithms, but also to reduce the dimensions of the data in order to achieve a less sparse result.

2.2.2 Distance functions

The second step in performing a cluster analysis to measure the distances between each data point. There are many different distance measures that can be used to perform this. Each distance measure handles the data and variables differently for their calculations, and the results can be very different. Therefore, it's important to choose a distance measure that is suitable for this specific instance and data, as choosing non-optimal distance measures that produce inadequate calculations [11].

Definition 1. A metric or distance function is a function $d(x, y)$ that defines the distance between elements of a set as a non-negative real number. [12]

Some examples of distance functions can be considered as:

- **Cosine similarity:** captures the orientation (the angle) of the documents and not the magnitude. If you want the magnitude, compute the Euclidean distance instead. [12]
- **Jaccard similarity:** is a measure of how two sets are similar. There is no "tuning" to be done here, except for the threshold at which you decide that two strings are similar or not. [12]
- **Normalized compression distance (NCD):** measures the distance by the difficulty of compressing one object into the other. [13]

2.2.3 Clustering algorithms

The third step in performing a cluster analysis is the actual clustering of the data. Like the distance measures, there are a lot of different clustering algorithms to choose from that performs differently, and depending on the purpose or idea of the clustering, it is important to have a clustering algorithm that is suitable for your approach. There are many ways to capture the notion of what a cluster means, that it's really hard to have one answer for everything. Different applications need a different notion of cluster [14].

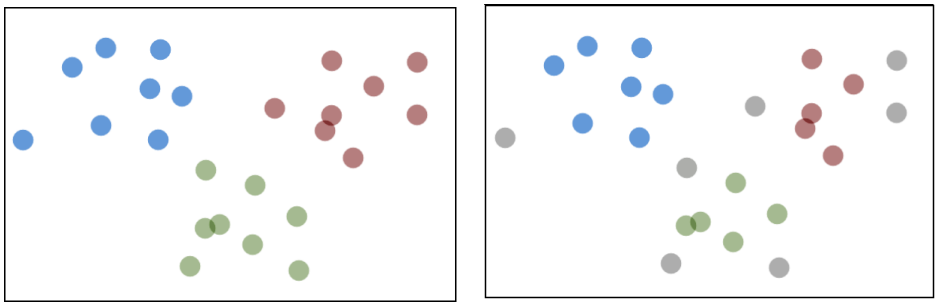
Clustering algorithms can be broken down into different groups. Centroid based, or parametric based, algorithms, requires parameters to operate, for example how many clusters are expected in the given data. This allows them to work very well on data that is small. On the other hand, there is density-based clustering, that is gonna allow your data to define the shape of your clusters. Density-based clustering is gonna require a lot more data to be effective at working [15].

Clustering can also be divided by flat versus hierarchical clustering. Flat clustering gives a single grouping or partitioning of your data, while hierarchical clustering returns sets of nested relationships between your data, like a tree representation where the branches can be used to extract the clusters [15].

Clustering algorithms can hence be divided into centroid based or density-based, and also flat or hierarchical.

An example of a flat centroid based algorithm is K-Means, which requires a given amount of clusters that are in your data to operate. This is only a wanted feature if you are familiar enough with your data to know how many clusters you expect. K-Means also forces all points of your data to belong to a cluster, which can cause very variable clusters in high dimensions as the data put in a cluster may not share many similarities to it at all [15].

An example of a flat density-based algorithm is DBSCAN, which tries to find areas in your data that are dense. DBSCAN uses a radius around each data point to find nearby data points, called epsilon. DBSCAN also allows data to be recognised as "noise", in other words data that doesn't belong to any cluster [14], [15]. Figure 3 shows an example of clustering data points where Figure 3(a) represents an example of clustering without noises and Figure 3(b) shows an example with clustering with noises.



(a) Clustering without noise (b) Clustering with noise

Figure 3: An example of clustering data points with and without noise.

An example of a hierarchical centroid based algorithm is Agglomerative Clustering. Since this algorithm doesn't know the number of clusters that it's working with, it's gonna build a natural hierarchy over the whole range of clusters and use that to try and divine the correct number. That is often going to be represented by a dendrogram. Figure 4 shows those clusters splitting up and breaking apart as it fragments down until you have every point of the data in their cluster. But again we get affected here by the parametric assumption of the number of clusters, and each data point is forced into a cluster regardless of it possibly being very different from it [14], [15].

HDBSCAN is an example of a hierarchical density-based algorithm that uses the same density algorithm as DBSCAN [14]. The difficult element of DBSCAN was finding the right value of the epsilon for the data: the radius of the circles around the point. Turns out that as epsilon decreases, we get a guarantee that components are only going to fragment into smaller clusters or vanish entirely. The mentioned fact allows the creation of a hierarchical tree-like the one used in the hierarchical centroid based algorithm Agglomerative Clustering, which is exactly what HDBSCAN does. HDBSCAN then simplifies the tree by eliminating any clusters with less than a given amount of points in them, and result in a much simpler dendrogram and then apply a cluster selection, like the process in Agglomerative Clustering) [14].

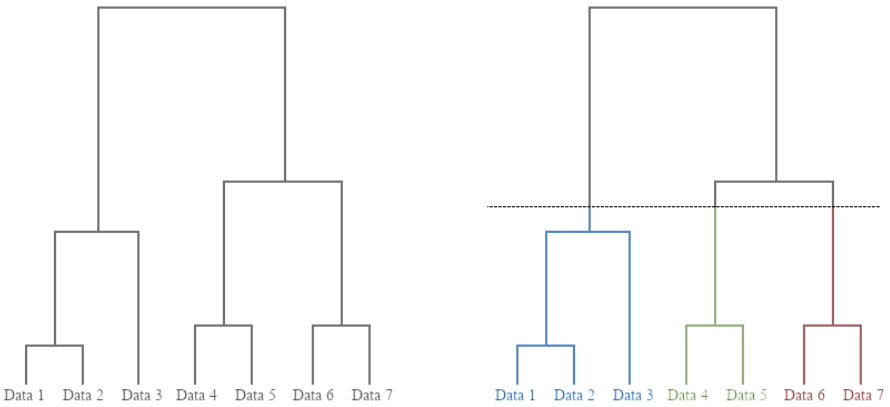


Figure 4: Example of how data clusters are split and selected up in Agglomerative Clustering.

2.2.4 The clustering performance assessment

The fourth and final step of performing a cluster analysis is to validate the result given from the clustering. Since cluster analysis is categorized as unsupervised machine learning, it is more difficult to know if the results are good or not. One common measure used in machine learning is the F-Score. The F-Score is the harmonic mean of the precision (the number of correct positive results divided by the number of all positive results) and the recall (the number of correct positive results divided by all samples that should have been identified as positive) of a test to calculate the score. A resulting score is a number between 0 and 1, where 1 is the best result indicating that both precision and recall were perfectly accurate. To calculate the F-score, as well as the precision and recall, it uses a binary classification with four basic combinations of the predicted outcomes and the actual true outcomes [16]. Table 1 represents the confusion matrix which allows visualization of the performance of an algorithm.

		True Condition	
		True Condition Positive	True Condition Negative
Predicted condition	Predicted Condition Positive	**True positive:** Predicted outcome is positive, True outcome is positive.	**False positive:** Predicted outcome is positive, True outcome is negative.
	Predicted Condition Negative	**False negative:** Predicted outcome is negative, True outcome is positive.	**True negative:** Predicted outcome is negative, True outcome is negative.

Table 1: Confusion matrix.

Thereby we can define precision and recall as:

$$Precision = \frac{\sum True\ positive}{\sum Predicted\ condition\ positive} = \frac{\sum True\ positive}{\sum True\ positive + \sum False\ positive} \tag{2.1}$$

and also:

$$Recall = \frac{\sum Truepositive}{\sum Condition\ positive} = \frac{\sum True\ positive}{\sum True\ positive + \sum False\ negative} \tag{2.2}$$

where the F-score is the harmonic mean of precision and recall:

$$F_1 = \left(\frac{2}{Recall^{-1} + Precision^{-1}} \right) = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{2.3}$$

3 Related Work

This chapter presents previous work and studies related to the thesis.

Test optimization has received a great deal of attention in the past decades because of its promising applications. Test cases will be prioritized and scheduled by the use of their dependency information to optimize the testing process. In this regard, many types of research are focused on this topic. In this chapter, we review and summarize some of the proposed related works in the area of test optimization.

Arlt et al. [4] demonstrated that overlooking or ignoring dependencies between test cases will lead to them failing after each other. In integration testing, dependency detection becomes even more important as the software modules are tested combined as a group. Indumathi et al. [17] proposed a manual method for detecting the functional dependencies between test cases through their sequence execution. However, their method was not optimal for a large set of test cases because of the dependency information being extracted manually. Caliebe et al. [18] introduced a component dependency model and proposed an algorithm that by analyzing the system structure, architecture, and requirements, could identify the dependencies between components in an embedded system.

Several types of research have been done regarding the application of Natural Language Processing (NLP) techniques in the domain of software testing, with specification mining and requirement engineering being the main focus [19]. Unterkalmsteiner et al. [20] studies how NLP can be applied for test case selection, Thomas et al. [21] propose a text similarity technique for test prioritization, and Flemström et al. [22] investigates an NLP approach for test automation prioritization.

Tahvili et al. [8] proposed a natural language processing-based approach that with test specifications given on integration level can discover test cases semantic dependencies automatically. The approach uses an algorithm (Doc2Vec) that converts each test case to a vector in the n-dimensional space whose similarity then can be measured by calculating the similarity between the vectors that represent the test cases. The algorithm performs the conversion with the help of deep language patterns based on sentence grammar. The vectors are then grouped to semantic clusters with the help of the HDBSCAN clustering algorithm that uses the distance between the vectors. Lastly, a set of cluster-based test scheduling strategies is proposed for execution. Their work showed that it is possible to discover semantic dependencies between test cases through analyzing test specifications and cluster test cases based on these semantic dependencies.

Tahvili et al. [9] investigated how functional dependencies between test cases can be discovered at an early stage of software development. This was done by analyzing the structure of requirement specifications for software with the help of natural language processing to acquire information relevant to the dependencies. Their work proved that it is possible to identify dependencies between test cases thought internal signals for in and out data that represent the test case dependencies. Test cases were classified as a dependant of each other if and only if the out data from one of the test cases is a requirement as in data to another test case.

Tahvili et al. [7] expand their previous research by introducing, implement and evaluate a natural language processing-based approach based on test cases similarities and differences from test specifications, to automatically detect dependencies between test cases. They wanted to use semantic similarity identification written in the test case specifications natural language without the use of test history, requirement specifications, or internal signals that they used in the previous research. In their approach, they once again used the Doc2Vec algorithm to discover semantic similarities between test cases and then group them with the help of two clustering algorithms (HDBSCAN and FCM). Their work showed, among other things, that it is possible to identify independent test cases and group test cases depending on each other into clusters, and the possibility of analyzing test cases written in natural language for a wide range of systems where test cases dependencies are common.

Feldt et al. [23] produced a model for test variation that was built on search-based software testing potential to increase the quality of test-related development activities and at the same time lower the cost. Their work focused on tests suitability about existing or previously found tests, compared to the previous research in search-based software testing, which primarily focused on the search for isolated tests that were optimal according to a fitness function guiding the search. In

correlation with the model, they proposed the usage of a theoretically optimal metric at variation points in the model, as well as a practically useful approximation to read the metric. The result showed to be powerful and could be optimized for several test measuring scenarios.

4 Method

This chapter describes the utilized techniques in order to complete the study and answer the research question.

The studies' research question will be answered through an experiment that tests the research question through implementation in order to handle the data that could be used in a real-life scenario. This is to achieve a result in the best way possible that can be used to optimize software testing procedures in practice. To show the viability of the proposed approach, Bombardier will prove a number of test cases in the integration test level of an industrial case study. Furthermore, the correct dependencies between the test cases will be provided that can be used for comparing the results.

The work is planned to be done in steps and starts with a feature selection of the data provided by Bombardier. This phase involves cleaning up the data by studying the test case files, identifying the relevant information, extracting that text that is then written into separated text files. Step two is to analyze the data by using Robert Feldt's framework extension [23] that uses a distance function to measure the similarity of the text files and collect the results in a distance matrix. After that, the distance matrix will be used for the clustering process by applying a clustering algorithm that creates clusters based on the values in the distance matrix. Lastly, the results given from the clustering will be compared to the correct dependencies in order to determine if the results are as anticipated and if the research questions are answered.

The distance function that will be used is the Cosine similarity, as it is one of the most popular similarity measures for text documents because it's the ability to consider repetition of word sets, rather than unique occurrences like for example the Jaccard distance [24]. The clustering algorithm that will be used is HDBSCAN due to it showing higher results compared to other clustering algorithms when applied to similar cases [7]. The result of the clustering will then be evaluated with the Ground Truth by calculating the F-Score.

5 Ethical and Societal Considerations

This chapter acknowledges the ethical and societal considerations taken into account in the study.

This study does not involve any experiments, researches, or approaches of other types that involve other people and therefore not manage any personal information. The goal of comparing the results of the automated process to a manual process is instead done through the correct dependencies provided by Bombardier rather than a testing group of people. Instead, the ethical focus lies on the aspects that Zobel [25] considers are important in correlation to scientific writing, which involves thoroughly separating facts from my personal opinions and to perform the study in an honest way - that is, to not misrepresent information, turning on the truth or plagiarize others work and present it as my own.

6 Industrial Case Study

This chapter presents the study's process of applying the chosen NLP techniques for clustering and validating the result. Section 6.1 gives a brief overview of the procedure. Section 6.2 describes the Ground Truth used to validate the result. Section 6.3 presents the process of the procedure in more detail.

In order to analyze the feasibility of the proposed approach in this thesis, we designed an industrial case study at Bombardier Transportation (BT) in Sweden, by following the proposed guidelines of Runeson and Höst [26] and [1]. BT provides multiple levels of manual as well as automated testing. The integration testing is entirely manually performed, and the large amounts of test cases required are done in multiple cycles of testing. Like mentioned earlier, knowing the similarities and dependencies between test cases can benefit from optimizing the process of the tests. This study analyzes the integration test cases for a project called BR490.

6.1 Unit of analysis and procedure

The units of analysis in the case under study are manual test cases at the level of integration testing for a safety-critical train control subsystem at BT. The case study is performed in several steps:

- An industrial testing project at BT called BR490 project is selected as a case under study.
- A total number of 1748 test specifications are extracted from DOORS and analyzed for text similarity detection.
- The Cosine distance function is applied to measure the similarity of the test cases.
- The HDBSCAN clustering algorithm is applied to group the test cases into clusters.

6.2 Ground Truth

A Ground Truth (GT) provided in [9] at Bombardier is employed to validate the obtained result in this thesis. This GT contains all the requirement labels, the label of each test case belonging to each requirement, and a list of requirement labels for each requirement that specified which requirements it is dependant of. A requirement can contain a single or multiple test case labels, and a test case label can belong to a single or several requirements. Each requirement can also be dependant on none, one, or several other requirements.

6.3 Empirical Evaluation

The test cases are studied and cleaned up by removing data that is not necessary for the calculation and clustering process, which can also slow the process and affect the results negatively. The information that is identified as relevant is then kept through the feature selection for further process, while the rest is removed. The test cases then need to be split into separate files. Since they were initially in a *csv* file, this can easily be performed by writing a macro in Microsoft Excel. The macro iterate through each row (each test case) and copy all columns on the rows (all data belonging to their respective test case) and save the data in a new file. The newly created files are named after their respective names provided in the test cases. These files are saved as *csv* files, in order to be used for the upcoming Cosine distance calculations.

The second step is to calculate the Cosine distances and similarities between the test cases. This is done through Robert Feldt's framework extension [23] that performs the selected distance function on the given set of files. The result is a distance matrix - a two-dimensional array with the size $N * N$, where N is the amount of elements used for the calculations. In this case, N is the number of test cases. The points in the matrix would represent the distances between the row and column element. In this case, the points represent the similarity between each test case.

The third step is performing the HDBSCAN clustering algorithm on the given distance matrix. This is done through creating a Python application with the required libraries that loads the

distance matrix, performs the clustering, and saves the results. The algorithm is set to allow noise in order to not force independent test cases into clusters. The algorithm is also set to allow a minimum cluster size of 2 in order to allow clusters with only a single dependency relation between two test cases. Otherwise these test cases can be classified as noise despite having a dependency, or end up as part of bigger clusters with smaller similarly to the rest of the cluster. The clustering result is provided as an array of N elements, where N is the number of elements in the distance matrix's lengths. In other words, the same amount of test cases used when creating the distance matrix. The values of the array represent the cluster-ID of each element, indexed in the same way as the distance matrix. Therefore, it is simple to pair up each test case to the respective cluster-ID.

The final step is to validate the results given by the clustering process. This is done by calculating the F-score by comparing the results from the algorithm with the dependencies in the Ground Truth using Equation 2.3. Two test cases are selected from a cluster and located in the Ground Truth, and then checked if they shared a dependency relation. This is repeated several times with randomly selected test cases from randomly selected clusters. Both expected dependencies as well as expected separations are tested in the validation. The result of each comparison is put in the respective category of the binary classification.

7 Results

This section provides an overview of the obtained results in this thesis. Section 7.1 presents the outcome of the clustering procedure. Section 7.2 presents the result from the validation process of computing the F-score by comparing the clusters to the Ground Truth (GT).

7.1 The clusters

Figure 5: Visualization of the clustered test cases using Cosine similarity distance and HDBSCAN.

Figure 5 shows a 2D visualization of the resulting clusters. Each point represents a test case and the distance between the points represents how similar or different they are from each other. Each unique color of the points represents the different clusters, where the gray-colored points represent noise and therefore not in a cluster. The number of 210 clusters is achieved, containing 750(42.9%) of the test cases. The other 998(57.1%) test cases are recognized as noise. The clusters vary in sizes between 2 and 16, with clusters of smaller sizes being the most common and bigger clusters being very few. Table 2 shows the number of clusters an also the cluster size.

Cluster size	2	3	4	5	6	7	8	9	10	11	13	14	15	16
# of clusters	78	52	42	4	7	3	2	1	2	1	1	1	1	1

Table 2: Amount of clusters of each size.

7.2 The validation

True Positive	72
False Positive	28
True Negative	86
False Negative	14
Precision	0.72
Recall	0.837
F-Score	0.7741

Table 3: The performance metric for the proposed solution in this thesis.

As mentioned before, the F1 Score metric is employed for the evaluation of the obtained results in this thesis using Equation 2.3. In fact, the F-Score of the resulting clusters when compared to the dependencies in the GT is presented in Table 3. The True Positives are the number of

comparisons where the test cases were correctly placed in the same cluster and rightfully shared a dependency in the GT. The False Positives are the number of comparisons where the test cases were wrongfully placed in the same cluster expecting a dependency that didn't exist. The True Negatives are the number of comparisons where the test cases were correctly placed in separate clusters or recognized as noise and rightfully weren't dependant on each other. The False Positives are the number of comparisons where the test cases were wrongfully placed in separate clusters or classified as noise despite actually sharing a dependency. The Precision is the number of correctly detected dependencies over the total number of detected dependencies using the proposed solution in this thesis. The Recall is the number of correctly detected dependencies over the total number of the existing dependencies in the GT. The resulting F-Score of 0.7741 tells us that there is a certain potential to discover test case's dependencies using this suggested approach to some degree, at least sufficient enough to make improvements in industrial settings.

Moreover, of the test cases that are clustered faulty, the following errors occurred:

- Some test cases that were found in the same cluster weren't dependant on each other as they should've been.
- Some test cases that were found in different clusters were dependant on each other as they shouldn't have been.
- Some test cases that had at least one of them classified as noise were dependant on each other as they shouldn't have been.

8 Discussion

This chapter highlights some of the threats to the study's validity and discusses the outcome of the approach. In Section 8.1, some future directions of the present work is suggested.

The result of the empirical evaluation shows that there is potential to discover test case dependency based on their semantic text similarity with the suggested approach. It should be noted that not all the test cases were validated, but only a randomly selected set. Therefore not all combinations of test cases were validated either. This would take an extremely long time to perform manually but could be done by creating a program to perform this automatically with accessing the clustering results and the Ground Truth. Validating all possible combinations of test cases would however result in a more accurate F-Score, as the few combinations that were randomly selected could have affected the resulting F-Score for better or worse.

Achieving perfect results, or results that are as close to perfect as possible would require some tweaking and testing. In order to know if the results are good or not, it would require some form of validation, like a Ground Truth to compare the result against like we had in this case. If a Ground Truth is already available, the automation process would be redundant as the dependencies are already discovered, and only useful for validating how accurate the automated results are. And even if a "perfect" setting would be found for that one scenario, it would most likely not be correct for another. This is important when applied in a real-life scenario, as even minor flaws might cause massive consequences depending on the system it's used in.

It may also be possible that achieving perfect results isn't possible with this approach. Studying the different settings and options for the algorithms, as well as several distance functions and clustering algorithms, would provide more information on how accurate results are reachable.

But how accurate does the result have to be in a real-life scenario? Are the perfect results required? Not necessarily, as independent test cases that are put in a cluster they don't belong in, wouldn't cause any scheduling disruptions, but rather just decrease the efficiency. However, test cases with dependencies that have been wrongly clustered could cause disruptions, and therefore have a much higher priority of being clustered correctly.

So even though Artificial Intelligence (AI) and Natural Language Processing has had a large impact on several aspects of software engineering and seems to show promising results in this area, it still needs further improvement to assure the quality that is still heavily dependent on human judgment and interaction. Therefore AI should be utilized solely for testing and research areas and not for practical implementations where faulty results of test case dependency determinations can lead to severe consequences. Using the results generated by AI along with human verification however, AI could potentially achieve an increasing involvement in software testing with even greater results in the future.

8.1 Future Work

The results of this study are only based on a single distance function and a single clustering algorithm. Comparing the results of different distance functions such as Jaccard or NCD combined with different cluster algorithms such as DBSCAN, Affinity Propagation or Agglomerative Clustering could yield interesting outcomes.

9 Conclusions

The goal of this study was to analyze the extent to which functional dependencies between test cases can be detected by analyzing their test specification. The study used a natural language processing (NLP) based approach that performed a cluster analysis on a set of test cases. After an initial feature selection, the test cases dependencies were calculated through a Cosine distance function to measure the similarities of the test cases. The result of the similarity calculation was then clustered using the HDBSCAN clustering algorithm. The clusters would represent test cases relations where test cases with close similarity were put in the same cluster as they were expected to share dependencies. The clusters were then validated with a Ground Truth containing the correct dependencies. The result was an F-Score of 0.7741.

References

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

[2] S. Tahvili, W. Afzal, M. Saadatmand, M. Bohlin, D. Sundmark, and S. Larsson, "Towards earlier fault detection by value-driven prioritization of test cases using fuzzy topsis," in *13th International Conference on Information Technology : New Generations (ITNG 2016)*, April 2016.

[3] S. Tahvili, M. Saadatmand, and M. Bohlin, "Multi-criteria test case prioritization using fuzzy analytic hierarchy process," in *The Tenth International Conference on Software Engineering Advances*, November 2015.

[4] 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 *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, April 2015, pp. 1–10.

[5] S. Tahvili, M. Saadatmand, M. Bohlin, W. Afzal, and S. H. Ameerjan, "Towards execution time prediction for test cases from test specification," in *43rd Euromicro Conference on Software Engineering and Advanced Applications*, August 2017.

[6] S. Tahvili, W. Afzal, M. Saadatmand, M. Bohlin, and S. H. Ameerjan, "Espret: A tool for execution time estimation of manual test cases," *Journal of Systems and Software*, vol. 135, pp. 1–43, September 2018.

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

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

[9] S. Tahvili, M. Ahlberg, E. Fornander, W. Afzal, M. Saadatmand, M. Bohlin, and M. Sarabi, "Functional dependency detection for integration test cases," in *The 18th IEEE International Conference on Software Quality, Reliability and Security*, July 2018.

[10] S. Tahvili, M. Saadatmand, S. Larsson, W. Afzal, M. Bohlin, and D. Sundmark, "Dynamic integration test selection based on test case dependencies," in *The 11th Workshop on Testing: Academia-Industry Collaboration, Practice and Research Techniques*, April 2016.

[11] P.-N. Tan, *Introduction to Data Mining*, 2nd ed. Pearson Education, 2019.

[12] S. Ontañón, "An overview of distance and similarity functions for structured data," *Artificial Intelligence Review*, pp. 1–43, 2020.

[13] R. Cilibrasi and P. M. Vitányi, "Clustering by compression," *IEEE Transactions on Information theory*, vol. 51, no. 4, pp. 1523–1545, 2005.

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

[15] A. Joshi and R. Kaur, "A review: Comparative study of various clustering techniques in data mining," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 3, no. 3, 2013.

[16] C. Goutte and E. Gaussier, "A probabilistic interpretation of precision, recall and f-score, with implication for evaluation," in *European Conference on Information Retrieval*. Springer, 2005, pp. 345–359.

[17] C. Indumathi and K. Selvamani, "Test cases prioritization using open dependency structure algorithm," *Procedia Computer Science*, vol. 48, pp. 250–255, 2015.

[18] P. Caliebe, T. Herpel, and R. German, "Dependency-based test case selection and prioritization in embedded systems," in *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*. IEEE, 2012, pp. 731–735.

[19] H. Hemmati and F. Sharifi, "Investigating nlp-based approaches for predicting manual test case failure," in *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2018, pp. 309–319.

[20] 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.

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

[22] D. Flemström, P. Potena, D. Sundmark, W. Afzal, and M. Bohlin, "Similarity-based prioritization of test case automation," *Software quality journal*, vol. 26, no. 4, pp. 1421–1449, 2018.

[23] R. Feldt, R. Torkar, T. Gorschek, and W. Afzal, "Searching for cognitively diverse tests: Towards universal test diversity metrics," in *2008 IEEE International Conference on Software Testing Verification and Validation Workshop*. IEEE, 2008, pp. 178–186.

[24] A. Huang, "Similarity measures for text document clustering," in *Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand*, vol. 4, 2008, pp. 9–56.

[25] J. Zobel, *Writing for Computer Science*, 3rd ed. Springer Publishing Company, Incorporated, 2015.

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