CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

An Empirical Study on the Trade-off of Diversity Types and Measures on Test Visualisation

Master's thesis in Computer science and engineering

Kamil Mudy
Andreas Månsson

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2021

Master's thesis 2021

An Empirical Study on the Trade-off of Diversity Types and Measures on Test Visualisation

Kamil Mudy
Andreas Månsson

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2021

An Empirical Study on the Trade-off of Diversity Types and Measures on Test Visualisation
Kamil Mudy
Andreas Månsson

Supervisor: Francisco Gomes de Oliveira Neto, Computer Science and Engineering
Examiner: Robert Feldt, Computer Science and Engineering

Master's Thesis 2021
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2021

An Empirical Study on the Trade-off of Diversity Types and Measures on Test Visualisation

Kamil Mudy
Andreas Månsson
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Test diversity is a well-researched topic and although much has been done in this area one part that is lacking is the investigation on how a functional dashboard can bring awareness to diversity into the software engineers' toolkit. To expand that toolkit we have performed several types of analysis and an experiment where we address, i) What are the trade-offs in diversity for using different diversity measures (e.g., Jaccard, Levenshtein, NCD) and types of test artefacts (e.g., input, output), ii) what are the visual and structural differences on the clusters after using different dimensionality reduction techniques (e.g., MDS, t-SNE, UMAP) and iii) to what extent do different visual techniques influence decisions about test case selection. Our key findings are that Jaccard and Levenshtein perform similarly and NCD performs worse as expected on smaller data. However, Jaccard runs faster which gives the edge to Jaccard. When it comes to dimensionality reduction techniques, we have found that t-SNE and UMAP have a better effect on the structure of similarity maps compared to MDS. Furthermore, UMAP has a slight edge over the other techniques because of its hyperparameters being easier to understand and performs faster. Finally, when it comes to developing dashboards that visualise diversity it is worth pointing out that the colour of the data has a big effect on decision making, even better if the colours are discrete and are coupled with a visual technique (such as making a test case's data point bigger if it found more faults). Additionally, we found that sliders and labels used to filter the data are not used much by developers when performing tasks on diversity visualisations. However, using these filtering techniques is recommended because they have the potential to spark discussion with people who are not as invested in the lower level details of the project (e.g., code, tests), such as managers, who possess more of a holistic view in most cases.

Keywords: Software testing, Diversity, Visualisation, Interactive visualisation, Experimental study, Dimensionality reduction.

Acknowledgements

First and foremost we want to thank Francisco Gomes for guiding and supporting us throughout the thesis as our supervisor. We also want to extend our gratitude to our families for supporting us throughout our education.

Kamil Mudy and Andreas Månsson, Gothenburg, June 2021

Contents

List of Figures

List of Tables

1
Introduction

Testing is used in the industry to maintain and develop quality software. In order to achieve greater test quality, testers often use many tools to help keep track and ensure that the tests they write are accurate, or that the test results can help in identifying and fixing faults. Many testing techniques offer benefits in improving test effectiveness, and diversity-based testing is one of them. Diversity represents how different objects from a set are from each other and the difference between those objects can be measured as a distance value. Visualising this distance, formally called information distance, between two objects we are able to see how diverse the objects are from each other. If diversity is not kept in check it can have negative effects on the entire test repository hindering test planning, maintenance and quality [1]. For instance, it could accumulate into redundancy and waste in test execution. An example of redundancy in test cases could be, two different test cases covering the same system component/requirement or method which in turn does not reveal any new information. Furthermore, visualising this diversity can trigger insightful conversations between stakeholders even if some might not have specific knowledge of the software.

Test diversity is a subject that is well researched and is the topic of many recent publications. Most diversity measures are classified as artefact or behavioural diversity [1, 2, 3, 4, 5]. Artefact diversity, mainly measures the distance between test artefacts related to the test specification such as inputs, outputs and test scripts, while behavioural diversity is mainly comparing the execution of test scripts and their behaviour such as failures and successes. Although the researchers have applied visualisation on test data to improve the comprehensibility of test artefacts, none really have investigated how a functional dashboard can bring awareness to diversity into the software engineers' toolkit.

Visualising data can be done with many different techniques. One of these techniques is a heatmap. Given any scale of numbers, usually zero to one, heatmaps will colour a grid depending on how high or low that value is on this scale of numbers. Heatmaps are especially good for showing intensity and clustering. Another visualisation technique is a similarity map [1]. A similarity map visualises the data in terms of similarity, i.e. if a data point is similar to another, the two data points will be close to each other on the graph and if they are not, they will be further apart. Similarity maps are useful for identifying clusters of outdated test cases or parts of the repository that require maintenance [1].

In order to make visualising the data useful and accessible to the users, the data could be presented on a dashboard. In software development dashboards are used to display key information to the users in a clear way. A familiar analogy is a dashboard in a car displaying all the relevant information about the car to the driver in a clear and easy to read fashion. However, software is not bound to the standards that are required in cars. In other words, when designing dashboards software developers have much more freedom over how they look and what they contain, meaning that we can tailor the key information presented, to the users need.

Dashboards can be composed of different visualisations and also different information. Diversity is one of such information. Visualising diversity shows benefits to testers [1], however, the distance (or similarity) between tests has many dimensions making it incomprehensible for humans. One way to mitigate the incomprehensibility of a multidimensional data structure is to use dimensionality reduction techniques. Dimensionality reduction compresses a multidimensional data structure from several dimensions down to a lower desired dimensionality. Most visualisations are plotted on two or three-dimensional planes (x,y or x,y,z). As such, we could use dimensionality reduction techniques to reduce any pair-wise comparisons (such as diversity) down to the desired dimension.

To further help us with visualising the data, clustering techniques are used on the data in combination with dimensionality reduction. The main objective of clustering techniques is to group together similar data based on parameters that feed into these clustering techniques. These parameters decide whether or not the data is close enough to be counted as a cluster.

1.1 Statement of the Problem

Although test diversity is a known and well-investigated topic, there are still some unexplored areas of this subject, one of them being the trade-offs between the combinations of different diversity measures and test artefacts such as test scripts, inputs and outputs (this is further discussed in the subsection below). In order to easily convey those trade-offs to industry practitioners, the data has to be visualised. Unfortunately, such data is usually composed of many dimensions and would be hard to read and visualise. Dimensionality reduction techniques together with clustering techniques are used in order to ease such visualisation. Finally, there exists a lot of different visualisation techniques and choosing between each one is a problem that poses some questions, for example, which visualisation technique conveys the information that we want the users/viewers to see in the best way. In summary, there are different, yet orthogonal problems here covering mainly (i) the diversity information chosen, (ii) the multi-dimensionality and comprehensibility of this information, and (iii) the visualisation techniques used to harness this information. Next, we detail each one of those three parts of our problem.

1.1.1 Diversity

There are many diversity measures that can be used to calculate diversity and it is not always clear which to use, e.g Normalised-compression distance (NCD), Jaccard, Levenshtein, Euclidean, among others. Most of the available techniques are limited in what type of information they can classify, such as strings or integers. Calculating diversity is also an expensive operation because it is a pair-wise operation. This means that the execution time for calculating diversity generally is $O(n^2)$ and in large test suits, this will take a long time to compute. The problem here is that determining the choice of measure is dependent on constraints from the information type and also the time available to run those techniques.

A solution for this is providing the testers with appropriate visualisations and in-depth analysis of each diversity metric and test artefact. By doing this we can display the trade-offs between each technique when it comes to time to perform and the accuracy of each metric.

1.1.2 Dimensionality Reduction

Diversity data as mentioned in the previous paragraph is multidimensional and hard to read for humans just by looking at it. Multidimensional reduction comes with several techniques, but the three main ones covered in this study are Multidimensional scaling (MDS), t-distributed stochastic neighbour embedding (t-SNE) and uniform manifold approximation and projection (UMAP). Although the three listed techniques are an upgrade to pair-wise comparison which is what is usually used in distance matrices, they are still quite technical and hard to grasp, since not everyone in the software industry is a statistical expert. The problem lies in knowing which technique to use for diversity measures in order to convey meaningful results to the tester such as waste in test repositories and efficiency in test execution. Since each of the dimensionality reduction technique gives a slightly different result in visualisation, it is important for the user to be able to distinguish between groupings of data or clusters.

A solution to this problem is to look at how each dimensionality reduction and clustering techniques impact the visualisations and determine which fits the situation best. As with the distance measures, there are different trade-offs to keep in mind, for example, the time it takes each technique to complete its dimensionality reduction, the retention of the original data structure and how "invasive" it is on the produced visualisation i.e. do those visualisations/plots change drastically depending on which one is used? Another thing to keep in mind is that dimensionality reduction techniques require different parameters and choosing these parameters is tricky for software engineers that are not very familiar with the topic of dimensionality reduction.

1.1.3 Visualisations

Dashboards aim to contain useful information about a certain topic/domain in an understandable and clear way. Especially when it comes to software testing which can be a complicated subject and which incorporates massive amounts of data, getting a grasp of the bigger picture might become overwhelming.

A solution to this problem could be achieved by using different visualisations such as heatmaps and similarity maps which already have shown some promise in research. According to de Oliveira Neto et al. [1], showing different kinds of graphs to the stakeholders in the industry trigger insightful conversations, support test maintenance and improved decisions. Having those kinds of visualisations on a dashboard could prove to make visualisations more accessible, support decision making by having the plots in one place and in turn lead to improved test case design.

1.2 Purpose of the Study

Our purpose with this study is to contribute to the scientific and technical knowledge about the visualisation of test diversity and the trade-offs between diversity metrics and test artefacts. We will achieve this by performing a study that will investigate the different diversity metrics, test artefacts, dimensionality reduction techniques with a clustering technique. Particularly, by studying the composition of the resulting visualisations. There will also be a technical contribution by developing a dashboard that will allow a user to see visualisations and the result of changing different parameters and better the understanding of how to visualise interactive diversity data.

1.3 Research questions

The proposed study sets out to answer three main research questions and their respective sub-questions presented below. In RQ1 we will focus on diversity metrics and the trade-offs between each of them when applying them to the test inputs and outputs. The answers that will be provided to us by RQ1 will be the inputs for RQ2 where we will focus on different dimensionality reduction techniques and clustering techniques and how they impact the composition and structure of similarity maps and data that is displayed on them. Finally, in RQ3 we will investigate how different visualisation techniques can help us convey relevant test information, such as waste or test efficiency, to the tester.

- **RQ1** What are the trade offs in diversity for using different diversity measures and types of test artifacts?
  - **RQ1.1** How do the distance values differ when using various diversity measures and types of test artifacts?
  - **RQ1.2** How do the distance matrices differ when using various diversity measures and types of test artifacts?

- **RQ2** What are the visual and structural differences on the clusters after using different dimensionality reductions?
- **RQ3** To what extent do different visual techniques influence decisions about test case selection?
  - **RQ3.1** Do different visualisations motivate testers to select similar / different sets of test cases for the same tasks?
  - **RQ3.2** What types of filtering, colouring and visual attributes are preferred when visualising diversity?

The paper is composed of multiple chapters. In chapter 2 we talk about the background and related work. We go into detail about diversity and distance measures, dimensionality reduction, clustering techniques and different types of visualisations as well as explanations of different terms that are present throughout the thesis. Chapter 3 presents our methodology where we go through the design and development of our experiments which produce our data to be evaluated. In chapter 4 we will present our results from the experiments detailed in chapter 3 and evaluate them. In chapter 5 we will discuss the results, write about the threats to validity and future work. Finally, in chapter 6 we will conclude the thesis.

1.3.1 Summary of Contributions

The scientific contribution of this thesis comes in the areas of investigating the trade-offs between different distance measures as well as looking into which dimensionality reduction techniques are best when visualising diversity. The thesis also contributes to the technical knowledge by performing a qualitative analysis on the topic of developing dashboards that help with the visualisation of diversity. Here we touched on topics related to the graphical elements of dashboards such as colouring of data points, filtering of the data etc. Gathering our results we made a small list of recommendations that a developer could follow to construct a dashboard that helps with visualising diversity.

In this thesis we found, as expected, that NCD found less nuance in the textual artefacts that were used in this study. However, we did find that NCD started performing better with much bigger data, almost equally to Jaccard and Levenshtein. Overall, we conclude that the distance function to use for input and output artefacts is Jaccard. It manages to capture nuance in even small data and it calculates the distance matrix faster compared to Levenshtein and NCD. In terms of input and output we could not find any differences between the two that would render an absolute answer, what was more important was the size of the data. When the data is too small, there is not enough possible variation and a small but significantly different input or output will still register as quite similar if it only shares some textual traits.

We also investigated the different dimensionality reduction techniques and found that UMAP and t-SNE both create visualisations that were deemed better in terms of clustering and overall structure. However, when it came down to UMAP and t-SNE they performed very similarly and there is no clear choice. The parameters

needed to run UMAP are far easier to understand than t-SNE's. Moreover, UMAP was quicker than t-SNE in our experiments. Therefore, we would recommend using UMAP over t-SNE.

While performing qualitative analysis on our developed interactive visualisations, which aim to help testers and developers with visualising diversity, we have found that using discrete colours together with other visual techniques (for example, the size of a test case data point grows bigger the more bug it finds) helps testers with selecting test cases to execute. However, using colour gradient is not preferred as it makes it harder to distinctively separate smaller differences. We found that interactive visual techniques, such as a slider or legend, could be used as a means to spark discussions around the diversity, but when it comes to the actual usage, we found that for developers, they do not bring much to the table and are rarely used while performing tasks related to which tests to execute in a test suit.

2
Background and related work

Testing is often used to evaluate the quality of software and is a vital part of a software project's life cycle. It is also true that testing requires large amounts of software development resources [6]. For this reason, it is important to make sure that running test suits is as optimal as possible in order to preserve resources such as time and money. One way to achieve this is by looking at statistics of a test suit, which consist of different type of test data, such as its diversity, time it takes to run the entire test suit etc and then taking appropriate measures based on the results.

In order to help to look at those statistics, visualisations help immensely as it is much easier for a human to look at pictures and make sense of them rather than only looking at the numbers. However, there are always some activities that need to be done along the way in order to make visualisation convey the relevant information.

In this thesis we will be investigating the diversity of different test repositories, visualising the results and finally evaluating them based on the visualisations. In order to do that there are some concepts that we need to look at which would help us with our topic. Concepts such as diversity and distance measures, dimensionality reduction, clustering and visualisation are described in more detail in the sections following this paragraph.

2.1 Diversity and distance measures

Diversity has been thoroughly researched in software use, specifically testing [2, 3, 1, 4, 5]. Diversity usually is investigated with the help of distance measures. The distance represents how similar (or diverse) a pair of tests are and is usually represented with a range from zero to one, where zero is identical and one is that they are dissimilar.

As previously mentioned in chapter 1, generally the distance measures that calculate diversity uses pair-wise comparisons. This means that each test case is compared to all other test cases (excluding itself), creating a nxn matrix where n is the number of test cases in a test repository. This matrix is also called a distance matrix and will contain all the distances between each test case in the test repository that is being investigated. The distance matrix is a data structure, where each column and row represents a test case, meaning that we can observe the distance between test case 1 and test case 2 in column 1 and row 2 as well as column 2 and row 1. Due to

this, we end up with two triangles, an upper and lower triangle, which both contain the same values.
The distance matrix is the underlying data structure that is used for the analysis that will be done in the following subsections as well as the input for the clustering techniques. In order to build the distance matrix, we need to choose a distance measure. There are many options but below we discuss a few which have been widely used in software testing.

2.1.1 Jaccard Index

Jaccard index was first introduced in the paper from 1901 by a Swiss botanist Paul Jaccard [7]. In the paper, he describes the jaccard index as the intersection between two sets divided by the union of the same two sets. In mathematical terms:

$$jaccardIndex(A, B) = \frac{A \cap B}{A \cup B} = \frac{A \cap B}{|A| + |B| - |A \cap B|} \tag{2.1}$$

where A and B are sets of q-grams which are pieces of strings, $A \cap B$ is the intersection between the two sets i.e. what both of them have in common and $A \cup B$ is the union of the two sets i.e. the combined contents of both A and B. In order to calculate the jaccard distance, the result needs to be subtracted from 1 as shown by the formula below.

$$jaccardDistance(A, B) = 1 - jaccardIndex(A, B) \tag{2.2}$$

Jaccard index has been widely used in previous empirical investigations of diversity-based testing [1, 8, 9, 10] Due to this we have determined that this should be one of the distance measures we will investigate.

2.1.2 Levenshtein Distance

Levenshtein distance [11] similarly to jaccard index is a string distance measure. The algorithm can be described in a simple way. Given two strings A and B, the Levenshtein distance is the lowest number of insertions, deletions or substitutions in order to transform string A to string B [12, 13]. This can be observed in the following example.

Consider the two strings "Tests" and "Case". The procedure for calculating the Levenshtein distance would be as follows:

- 1. Substitute 'C' with 'T' in the second string
- 2. Substitute 'a' with 'e' in the second string
- 3. Substitute 'e' with 't' in the second string
- 4. Insert 's' in the second string

Hence the Levenshtein distance will be 4 between these two strings.

Levenshtein is widely known and used in many different disciplines such as language studies [13] and mathematics [12] and more. It is also used in the field of software

testing [14] and as such, due to its popularity and usage within software engineering, we have chosen to include Levenshtein as one of the distance measurements to investigate.

2.1.3 Normalised Compression Distance

Compared to the two aforementioned distance measures, Normalised compression distance (NCD) which was first introduced by Li et al. [15] is not strictly a string-based distance metric, but rather compresses the data and works at the binary level. NCD is based on information distance which in turn is an extension of Kolmogorov complexity [16]. Kolmogorov complexity of an individual object is the length of the shortest binary program that computes said object i.e. given a string x Kolmogorov complexity, $K(x)$, is the length of a binary program that returns such a string [17]. Information distance, in turn, is as said previously an extension of Kolmogorov complexity where instead of measuring the absolute length of a single finite object it measures the absolute distance between two finite objects [16]. Adding on to that, Normalised Information Distance (NID) unlike information distance measures the relative distance between two objects [15]. Following we can observe the formula for the normalised information distance and how it extends the Kolmogorov complexity.

$$NID(x,y) = \frac{max\{K(x|y), K(y|x)\}}{max\{K(x), K(y)\}} \tag{2.3}$$

Kolmogorov and NID are both uncomputable, hence the development of NCD which uses compressors to approximate Kolmogorov complexity [18]. In the following equation we can observe the similarities it shares with NID, where $C()$ is the compressor used to approximate $K()$ and the better the compressor is, the better approximation of NID you will get.

$$NCD_C(x,y) = \frac{C(xy) - min\{C(x), C(y)\}}{max\{C(x), Z(y)\}} \tag{2.4}$$

Due to NCD being universal, it can be applied next to anything, which makes it a popular diversity measurement. It is used in several studies and has shown great results [4, 5, 3, 19].

Hernard et al. [19] shows that test case prioritisation using diversity information from NCD shows promising results. They show the effectiveness by calculating APFD (Average percentage of fault detected) with the test case prioritisation with the use of NCD diversity values among others and compares different techniques.

Feldt et al. [3] discuss the use of a practical test diversity metric and proposes the use of Normalized compression distance (NCD), since it conveniently works on all types of input, due to it being a data compression technique. In this paper, they conclude that they are able to cluster test cases by cognitive similarity, which implies that humans are able to understand and make sense of the information.

As such, given this information, NCD will be the final distance measurement that we will investigate.

2.2 Dimensionality reduction techniques

Multidimensional data is hard to understand, luckily there are ways to reduce data to fewer dimensions. We want to reduce the data to be able to understand, manipulate and interpret the underlying information that exists in the large scale of data produced by the diversity measurements. Reducing the dimensionality of the data into a more meaningful dimension will allow us to visualise the data with minimal information loss [20], to further analyse the data. Optimally we want to reduce the data to two dimensions since the visualisations we are using are composed in 2D space (x-axis, y-axis). One technique to reduce dimensionality can be observed in the paper by de Oliveira Neto et al. [1], where the authors use multidimensional scaling (MDS) for visualising diversity.

MDS is able to reduce the complexity of data but still preserve the distance data for visualisation. Another more state of the art technique for dimensionality reduction is t-distributed stochastic neighbour embedding (t-SNE). Tahvili [21] proposes the use of t-SNE to visualise density range in clusters created with machine learning. A third dimensionality reduction technique appropriate to our thesis would be uniform manifold approximation and projection (UMAP). UMAP was recently published by McInnes et al. [22], in which they state that UMAP is competitive with t-SNE in runtime performance and preserving data structure.

Becht et al. [23] proposes a comparison between UMAP and t-SNE where the results indicate that UMAP performs similarly on visualisations, but in regards to other aspects such as speed and preserving data structure, UMAP outperforms t-SNE and seems to be best all-around choice. The three different dimensionality techniques that we went through here, are all suitable for reducing the dimensionality of our data, due to all of our data representing diversity, which is a measure of difference that yields a multidimensional output.

The following subsections go more into detail about each of the dimensionality reduction techniques that are used in this thesis.

2.2.1 Multidimensional Scaling

The theory and methodology of MDS have been introduced by Torgerson [24] where he describes MDS as involving three basic steps. The first step is to gather comparative distances between each pair of data. The second step is converting the comparative distances into absolute distances with the help of estimations and the third step entails projecting the data to lower dimensional space while keeping the values as true to the original as possible.

According to Cox and Cox [25] there exists two different versions of MDS - metric and non-metric. Metric MDS, which is also called Classical MDS, works on an input distance matrix and performs a linear function to map the dissimilarities of each data point in the distance matrix. Non-metric MDS does not have any constraints

towards linearity and can thus use any positive monotonic function [26]. In this thesis, we will use Metric/classical MDS, due to the fact that we are using raw distances as data, whereas if we would have an ordinal scale, we would use non-metric.

MDS is a tried and tested technique that has been used in many different domains successfully but suffers from one big problem according to Van Der Maaten, Postma, and Van den Herik [27]. The problem that classical MDS suffers from is that it mainly focuses on preserving the larger distances that are generated by the pair-wise comparisons instead of the smaller distances. The smaller distances are equally important when looking for clustering and similar data that are constructed by their close proximity.

MDS has been used for visualising test diversity before [1] and due to its simplicity and popularity, we have chosen to include MDS as one of the techniques we will investigate.

2.2.2 T-Distributed Stochastic Neighbour Embedding

T-Distributed Stochastic Neighbour Embedding (t-SNE) was first introduced by Van Der Maaten and Hinton [28] which is a variant of the originally proposed Stochastic Neighbour Embedding by Hinton and Roweis [29]. t-SNE's objective is similar to MDS, where it reduces the dimensionality of a data set from high dimensionality to two or three-dimensional space which can then be visualised. The authors of the dimensionality reduction technique in question say that it produces significantly better results on visualisation than the SNE version, as well as other dimensional reduction techniques that were available at that time and it is much easier to optimise [28].

T-SNE is a technique that relies on machine learning in order to produce its work, and as such requires some hyper-parameters in order for the algorithm to work effectively. The algorithm is divided into cost function parameter and assisting optimisation parameters both of which are set manually. The cost function parameter is perplexity and the assisting optimisation parameters include the number of iteration, learning rate and momentum [28]. In order to use the algorithm to its full potential and produce interpretable visualisations, the user needs to tweak its inputs depending on the size of the data that is being fed to the algorithm. The importance of making sure that the hyper-parameters are set up correctly is presented by Wattenberg et al. [30] where he goes through the effects that each hyper-parameter has on the data.

Although t-SNE has many positives it also suffers from a few drawbacks which the authors have listed in their paper [28]. The first drawback of the t-SNE algorithm is that it doesn't perform well if the dimensionality needs to be reduced to a number that is higher than three. Secondly, there exists an issue of intrinsic dimensionality also know as the curse of intrinsic dimensionality. Intrinsic dimensionality refers to the amount of parameters needed to describe a multidimensional data set [31]. The authors of t-SNE specifically mention that their algorithm might not perform well

with data that has very high intrinsic dimensionality such as images due to it being based on local properties of data [28]. The final drawback of t-SNE is that the cost function parameter (perplexity) of the algorithm is not convex which in turn leads to the algorithm needs a couple of optimisation parameters which are mentioned in the previous paragraph.

T-SNE has been used for visualising test diversity before [21] and is one of the more popular dimensionality reduction techniques and as such we have decided to include t-SNE as one of the dimensionality reduction techniques to investigate.

2.2.3 Uniform Manifold Approximation and Projection

A recent addition to the dimensionality reduction techniques is Uniform Manifold Approximation and Projection (UMAP) [22]. The theoretical aspects of this technique are based on manifold theory and topological data analysis, but we will not go into more detail about these concepts in this paper but refer to the citation made at the beginning of this paragraph which will lead to appropriate sources if the reader wishes to read more about it, however, it is important to explain roughly how the algorithm works.

UMAP similarly to t-SNE is classified as a k-neighbour graph-based algorithm. This means that practically it can be described in terms of construction and operations on weighted graphs [22]. A weighted graph is a node branch type of graph where each node-connecting branch is given a value. UMAP similarly to other dimensionality reduction techniques of this type works in two phases, graph construction which is where the k-neighbour graph is created and graph layout where the lower dimensional layout of the graph is produced [22].

In order to use the UMAP algorithm correctly, a few hyper-parameters need to be kept in mind. They are, as stated by McInnes et al. [22] as follows: x data set that the dimensionality reduction will be performed on, n neighbourhood size, d amount of dimensions to be reduced, *min_dist* which controls the layout i.e. the spacing of the data points on visualisation and *n_epochs* which is the amount of times that the algorithm will be trained on the provided data set. Depending on the composition and the size of the data set as well as the intended dimensional reduction the above hyper-parameters will need to be adjusted accordingly.

As with t-SNE, there exists a couple of drawbacks to using the UMAP algorithm. The authors mention its interpretability as the first problem and refer to using linear techniques if interpretability is of critical importance to the user. The second problem that is mentioned is that given a smaller data set UMAP might produce a lot of false positives i.e. it can find structure in the noise of the data set. This means that UMAP falls short on data that is smaller in size. The third problem that the authors of the algorithm mention in their paper is that UMAP is not great with capturing the global structure of the data set but works well with the local structure.

UMAP is a relatively new dimensionality reduction technique that is mainly used in scientific fields such as biology [23], it has yet to make a firm appearance in software engineering, even though it has seen positive results in comparisons to t-SNE. Due to its novelty and reported positive result, we have decided to include UMAP as the last technique for dimensionality reduction in our report.

2.3 Clustering techniques

Clustering techniques or cluster analysis generally refers to the act of subdividing a set of data into smaller sets of data where the data points are pair-wise similar [32]. Clustering techniques such as the ones from Tahvili paper [21] or ones from Becht et al. [23] are usually used with the above mentioned dimensionality reduction techniques. From the combined results that the clustering and dimensionality reduction techniques produce it is possible to make a visualisation in which we can divide data points into different clusters with the help of colours, which could then be used for cluster analysis. An example of this can be seen in the figure. 2.1, where we have three distinct clusters, where each cluster is represented by a different colour.

Using clustering with diversity is not a novel concept and makes for excellent analysis in terms of cost-effective testing as presented by Tahvili et al. [21]. The thought behind it is that if there is a big cluster of test cases, you know there is a lot of test cases that cover the same aspect of the software or perform similar actions. This is not a perfect scenario when thinking about cost-effectiveness, where you want to minimize the time taken (cost) to run the test and maximize the coverage (effectiveness) of the test suit.

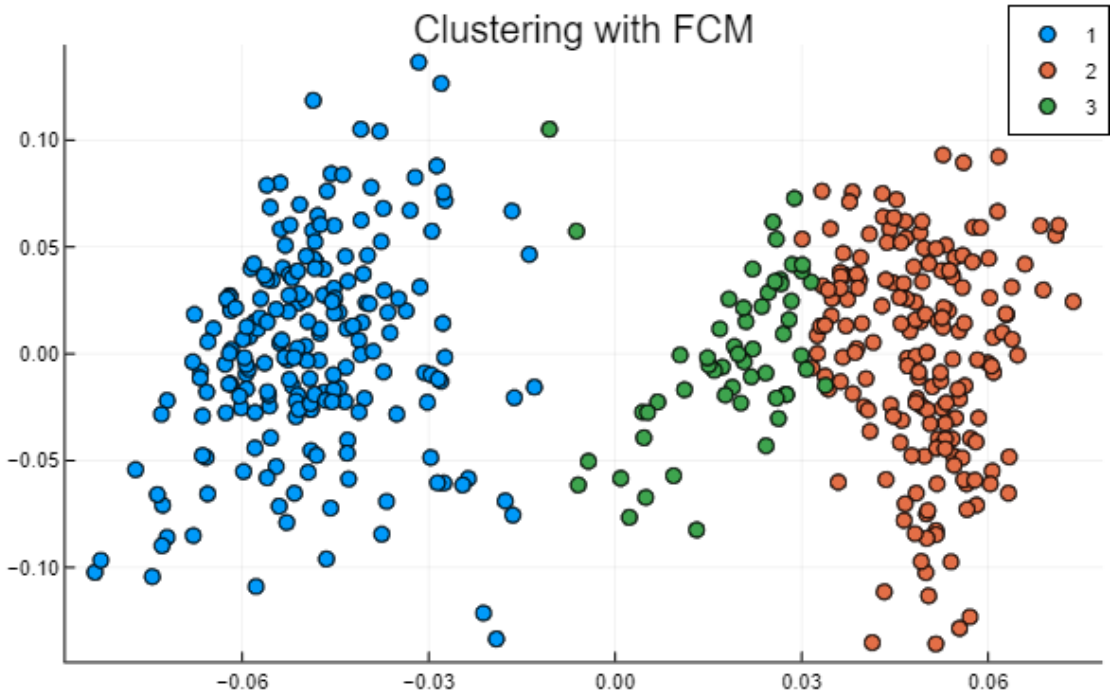

Figure 2.1: An example of clustering using test case artefacts from an example test repository and Fuzzy C-Means as clustering technique

2.3.1 DBSCAN: Density-based spatial clustering of applications with noise

A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise (DBSCAN) was first introduced by Ester et al. [33] and was designed to discover clusters of any arbitrary shape in data. The algorithm requires only two parameters *eps* which is the radius of the point neighbourhood i.e the radius of the cluster and $MinPts$ refers to the minimum amount of points around a specific data point to form a cluster. The inputs to the algorithm similarly to the dimensionality reduction techniques largely depend on the size of the data. The authors suggest starting with $MinPts$ value of 4 for two 2 dimensional data and claim that higher values require more computation and the results are not that different [33].

DBSCAN according to Hahsler et al. [34] is a well studied and widely used clustering algorithm in the scientific community. Due to its popularity and high usage, we think it is fit for us to include and analyse DBSCAN as one of the clustering techniques we will investigate in our report.

2.4 Visualisation

Visualisation has already shown that it could improve the comprehensibility of different topics contained in software engineering [35] by the use of different tools that

it has to offer such as graphs, plots, images etc. In software testing, visualisation has also been proven to be effective at clearly and easily conveying hard to grasp information to the targeted recipients of the study [36]. Although successful in conveying information, visualising seems to be more effective on systems of smaller scales as opposed to systems that are growing in size, this is because software visualisation solutions try to fix the problems at hand instead of thinking forward [37, 38]. Additionally, adding an interactivity layer can enhance visualisations by allowing software engineers to change different attributes of the charts (e.g., increasing size, adding/removing graphical elements, changing colours) to compare different states of the data shown [39].

To bring up a specific example of visualisation, we can look at the paper written by Feldt et al. [40], where the authors critically evaluate heatmaps and conclude that even a simple heatmap can bring a lot of valuable information to the tester such as understanding development and testing behaviour and analysing patterns. Furthermore, the authors mention that in order for a heatmap to be truly useful it needs some kind of interactivity such as support for filtering, grouping of items etc.

Connecting this to the dashboards which we would ultimately want to develop and which deals with a more specific aspect of software testing, namely testing diversity, visualisation has also been used to a great effect. Evidence suggests [1, 40] that using different visualisations on test diversity has a positive impact on testers, where it can lead to a better decision with regards to their testing suits as well as spark discussions around the results potentially leading up to greater test suit optimisation.

3
Methods

To answer our research questions which are briefly outlined in the introduction to this thesis, we decided to take the experimental study approach as our research method. In order to plan, execute and report our experiment, we use the guidelines proposed in the book by Wohlin et al. [41]. For research question 1 we were aiming to do a full factorial design, but due to time constraints, we decided to settle for a fractional factorial design. For research question 2 we will perform a visual analysis in order to evaluate the structural differences between the different dimensionality reduction techniques. For research question 3, however, we decided to go with a qualitative analysis, in order to evaluate the produced results and complement our experiment.

In summary, the goal of our experiment will be to take the results from RQ1 put them as an input to RQ2 and finally perform some qualitative analysis which would complement our experiment and help us answer RQ3.

3.1 Scope

In order to define the scope of our experiment, we need to think about theoretical and observational constructs that lay the foundation for the experiment. At the theory level, we want to see how does the diversity of different test suits impact the composition and structure of their visualisations. By creating these visualisations we want to turn the attention of testers/developers to the time-cost effectiveness (e.g., manual test selection) of their test suits and improve the way we visualise diversity. In turn, our observable constructs are that different diversity measures in combination with dimensionality reduction and types of test artefacts affect the visualisation of diversity in similarity maps and heatmaps. To further simplify our goals we summarise our scope according to the guidelines suggested by Wohlin et al. [41].

Analyse the **different types of test diversity**
for the purpose of **interpretabilty**
with respect to their **effect on test visualisations**
from the point of view of the **tester**
in the context of **open source project**

3.2 Planning

In this section we will describe our research questions in much more detail and explain the set-up of our experiment, describing all the factors and levels as well as how the experiment connects to each research question. Technical details, data and tools used in our experiments are described in section 3.3.

3.2.1 Research Questions

- **RQ1** What are the trade offs in diversity for using different diversity measures and types of test artifacts?
  - **RQ1.1** How do the **distance values** differ when using various diversity measures and types of test artifacts?
  - **RQ1.2** How do the **distance matrices** differ when using various diversity measures and types of test artifacts?
- **RQ2** What are the **visual and structural differences** on the clusters after using different dimensionality reductions?
- **RQ3** To what extent do different visual techniques **influence decisions** about test case selection?
  - **RQ3.1** Do different visualisations motivate testers to select similar / different sets of test cases for the same tasks?
  - **RQ3.2** What types of filtering, colouring and visual attributes are preferred when visualising diversity?

In RQ1 we are investigating the different combinations of artefact types and diversity measures to perform analysis on the resulting distance matrices. This is done in order to find the most suitable combination of artefact types and diversity measures. In RQ1.1, we will aggregate the separate distance values, for example, Jaccard and Levenshtein to further analyse the differences between the diversity types and measures. In RQ1.2 we will analyse distance matrices as a whole and compare them to see if there is a correlation between distance matrices created using varying distance measures. This can be done with a Mantel test to look for correlations between the distance matrices of several test suites. In addition, we will compare whether those correlation scores have an impact on typical applications of diversity such as test case prioritisation. In other words, we want to see how the different combinations of test artefacts and diversity measurements impact the test case ranking. This will provide us with the more suitable combinations of each diversity measurements and test artefact types.

In RQ2 we will use different dimensionality reduction techniques and a clustering technique on the obtained result from RQ1. We will analyse the application of these dimensionality reduction techniques into the different diversity functions by looking at the structure of the resulting similarity map as well clusters. By comparing the different visualisations between all the investigated projects, we are able to visually analyse the differences between the dimensionality reduction techniques.
In summary, we want to distinguish the trade-offs in each dimensionality reduction

by analysing the resulting similarity maps.

In RQ3 we apply interactivity and different kinds of filtering to our resulting similarity maps in order to see how do the testers perform tasks and work with visualisations that connect to test diversity. Therefore, we create three different use cases that represent tasks that a tester would do for test selection. We designed variations of the same dashboard with different visualisation techniques (e.g., filtering, colouring, layout) and, through an interview study, we verify how practitioners use those different visualisations to solve the tasks. For example, is filtering useful when trying to select different subsets of the available tests, such as 20%, 40% of the test suite, etc. This will allow the tester to see which clusters are removed /remain when the filtering is applied and, consequently, the perceived redundancy between tests that belong to the same cluster. We perform a qualitative analysis to answer RQ3 which entails coding and interpreting our interviews with subjects (e.g., students or practitioners in software engineering).

3.2.2 Experimental variables

Table 3.1: Factors, levels and their connection to quantitative research questions.

Factor Label	Factor name	Levels	Connection to RQ
F1	Diversity metrics	NCD, Jaccard, Levenshtein	RQ1
F2	Diversity types	Input, Output	RQ1

The table 3.1 describes the factors and the levels as well as the connections each one has to the research questions.

For **RQ1** we will have two different factors, which will be different diversity metrics and different diversity types. The three levels for factor one will be NCD, Jaccard and Levenshtein while the levels for factor two will be input and output (see table 3.1). The two dependent variables will be, distance matrix and distance values. This is due to the fact that we want to see how the different diversity metrics and diversity types will affect the results in each distance matrix and more closely its values (e.g mean distance, variance in differences per tests case). Then, we observe which combination of diversity types and metrics provide the most varied distance matrices to allow the measure of the correlation between those matrices. In short, the null hypothesis for this research question is *There is no difference between the different diversity types and measures in the test prioritisation rankings.* We measure those differences in relation to fault detection and priority of the test cases (i.e., ranking).

In **RQ2** we perform a visual analysis of different dimensionality reduction techniques, namely UMAP, t-SNE and MDS. We will look at the resulting layouts and

how each dimensionality reduction technique constructs the visualisation. Additionally, we look at the distribution of data points and clusters and how each technique visually deals with potential anomalies in the data if there exists any. Finally, we look at the composition of those clusters, e.g., whether the data are very spread out versus data that is clustered. Since this will be a visual analysis, RQ2 is not included in Table 3.1

In **RQ3**, we aim to perform a qualitative analysis of answers from interviews with subjects performing the designed tasks and reporting on the dashboards used. Therefore, we did not include RQ3 in Table 3.1.

The controlled variable for the experiment is open sources projects such as parsers and various GNU programs which can be found on Github or other repositories. We describe the specific programs in the instrumentation and execution section of this chapter.

3.3 Instrumentation and Execution

Below, we describe the different programs implemented and technologies used in order to perform different actives within the experiment, such as collect data or run the experiment. Figure 3.1 shows the steps of execution of our experiment in connection to the research questions.

To perform our experiment we developed a script in Julia programming language to run the most important tasks in our execution such as diversity metric calculation, dimensionality reduction, clustering and data visualisation. To compliment our main script we developed a couple of Python3 scripts which performed data scraping, analysis of distance matrices (mantel test), statistical analysis and the visualisation dashboards for research question 3.

Julia packages:

- MultiDistances.jl - version: 0.1.4
- StringDistances.jl - version: 0.5.1
- TSne.jl - version: 1.2.0
- UMAP.jl - version: 0.1.8
- MultivariateStats.jl - version: 0.8.0
- StatsPlots.jl - version: 0.14.19
- Clustering.jl - version: 0.14.2

Python packages:

- Pandas - version 1.2.1
- plotly - version 4.14.3
- scikit-bio - version 0.5.6
- scikit-learn - version 0.24.1
- scipy - version 1.6.0
- numpy - version 1.20.1

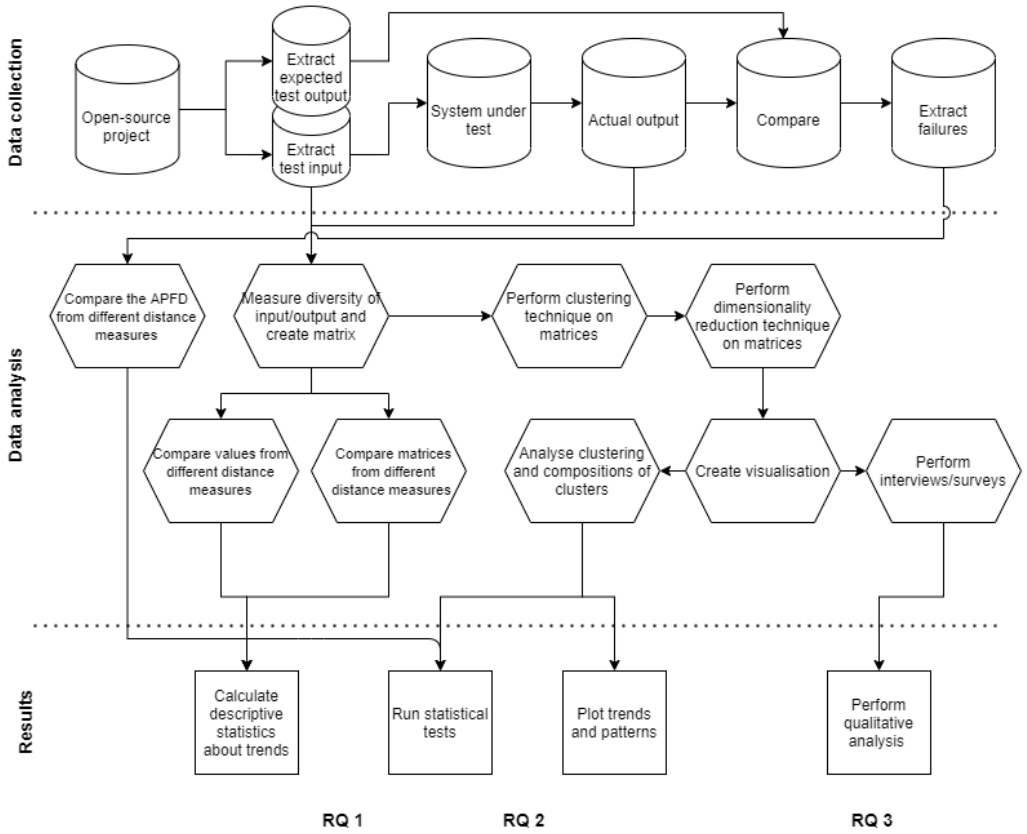

Figure 3.1: Steps taken in the execution of our experiment

We run out experiments on a PC with an Intel i7 CPU (4790k 4.0GHz), 16 GB RAM and a GeForce GTX 1070 GPU. We used Windows 10 (Education 64-bit) as the operating system. Nonetheless, the instrumentation uses Python and Java which work on various platforms.

3.4 Projects and Data Collection

The data that is being used for the experiments have been collected from three open-source projects extracted from research repositories. The selected projects are NanoXML, Sed and Grep. Those three projects have been used extensively in previous software testing research and are accessible in open repositories. Moreover, the projects contain useful information for research, such as fault detection via mutation scores, coverage, test and source code, build scripts, etc. NanoXML and Sed were extracted from the SIR (Software-artifact infrastructure Repository) repository[1], whereas we use the replication packaged offered by Henard et al. [19], to run the experiments with Grep[2]. Next, we detail each project and they have been used in our experiment.

Grep: Grep is a small (10068 lines of code (LOC)) program to search through a

[1] `https://sir.csc.ncsu.edu/portal/index.php`
[2] `https://henard.net/research/regression/ICSE_2016/`

file or other input streams in order to find matches based on a search string query and the file to be searched as a command-line argument. The grep program is well studied and described by the authors of the paper written by Henard et al. [19]. The repository created by the authors of the paper contained 5 different versions of the Grep program with 440 sample input test cases and corresponding test outputs as well as a varying number of mutants per version. A mutant in this case is a fault that is introduced to the Grep program. A test case input consisted of a program name (grep), a set of flags, a regex search expression and the search target which, in this case, was a set of scripts containing the logic of the Grep program. The output file was composed of the matches made by the regex expression. Out of the 5 versions of the Grep program we decided to focus on version 1 (56 mutants/faults), because of time constraints of the thesis.
Sed: Sed is a stream editor in which you can read an input stream (files or pipes) and perform changes to the text. Sed is a Unix utility that still to this day is available with most operating systems. The repository we used for Sed was the SIR repository [42], which hosts different versions of Sed with seeded faults, input and output data for everyone to use and perform experiments with. It is quite a small piece of software (14427 LOC) which is written in C. The inputs consist of different Sed commands which are performed on a single file, while the output is a file that saves the actions performed by the commands.
NanoXML: NanoXML is a small XML parser that is written in Java which can be easily run without the need for external libraries. It is fairly small with 7646 LOC, 24 classes. NanoXML contains documented seeded faults and five different versions that we can run. The inputs for this program are extensible markup language (XML) files and the produced output depends on what you are parsing. It can give either the parsed result in a different file format, or it can produce another XML file with the parsed result.

3.4.1 Interviews

In order to gather data for the RQ3, we conducted 5 semi-structured interviews in which practitioners were shown different versions of a similarity map (e.g., different filtering options, colour scales, etc.). We used RQ3 and its sub-questions as well as three use cases to structure our interview. The three use cases used to illustrate different test selection tasks are:

- Task 1: Chose five tests to execute based on the visualisation. Since the similarity map displays clusters of tests, the goal in task 1 is to investigate which information helps testers decide how to sample from those clusters.
- Task 2: Find three test cases to debug in order. We used different types of shapes and colour scale to convey the number of faults revealed by each test. Therefore, we aim to identify which option helps practitioners deciding where to begin debugging those shown faults.
- Task 3: Find three test cases that reveal bug 8 & bug 1.[3] Lastly, we allow practitioners to use different options to filter test cases based on their coverage

[3]Bug 8 and 1 were two arbitrary faults chosen from our projects.

Table 3.2: Interview participants. Experience is measured in years in their current industrial or academic position.

ID	Role	Experience (Years)
Person 1	Software Engineer	2
Person 2	Software Engineer	2
Person 3	Software Engineer	8
Person 4	PHD Student	2
Person 5	Masters Student	2

of specific bugs. The goal is to see whether a specific type of filtering supports prioritising tests based on fault coverage.

During the interviews we allowed the participants to be descriptive when answering questions and interacting with the dashboards. We also notified the participants that there is never a right or wrong answer to make sure that they don't feel under pressure and can solely focus on giving us their real opinion about using the different visualisation options presented.

Participants: The participants that were involved in our study mainly came from our academic and industrial networks. They span from senior to junior software engineers and students at masters or doctorate level with previous work and academic experiences within the area of software engineering. The overview of the participants is shown in Table 3.2.

Protocol: The interviews were conducted over a period of two weeks. Each interview lasted on average 45 minutes and was conducted through a video conferencing application called Zoom. Face to face interviews was not possible because of the COVID-19 pandemic. We asked the participants for their permission to record and to use the data in our study before each interview. The participants were also given an option to opt-out of the study at any time during the interview but none did. During the interview, we presented the participants with 3 sets of visualisations, each set for one of our three tasks. The first and the third set of visualisations contained 2 similarity maps and the second set contained 3. Each similarity map was exported as a standalone HTML file and sent to participants via a link to be used only during the interview. Therefore, participants could interact live with the maps during the interview. Each similarity map differed slightly in terms of presentation of data, such as different interactive filters, different colour scales, jittering and alpha turned on or off.

The flow of working with the visualisation in the interview was as follows: first, a participant was given an explanation and a demo about the similarity maps explaining what information they convey and the different interactive controls (e.g. zooming, panning, resetting axes). Then the participant was given a background scenario and the corresponding task. The participant had to use the similarity maps

Table 3.3: Interview questions.

ID	RQ	Question
—	—	**Information about Participants**
0	—	Can you describe your job title and responsibilities at work?
0.1	—	Do you have experience within software engineering and if so, do you have experience within software testing?
0.2	—	What types of software testing are you familiar with or have you been using, either in your company or prior experiences?
0.3	—	Have you used any sort of tools for test visualisation or analysis?
—	—	Intro to our thesis and what it is about.
—	—	Present slide for diversity input (BMI)
0.4	—	Do you think of diversity when creating tests?
0.5	—	Do you distinguish between input and output when constructing tests?
—	—	**Demo use-case 1**
1	—	Task: Chose five tests to execute based on the visualisation.
1.1	3.1,3.2	Which of plot A and B do you find more useful and why?
1.2	3.1	Would you rather see ranking to be percentage based or individual ranking for each test case - as it is right now?
—	—	**Demo use-case 2**
2	—	Task: Find three test cases to debug in order.
2.1	3.1, 3.2	Which of plots A, B and C do you prefer and why?
2.2	3.2	Would you prefer the visualisation to have jittering and/or alpha, or without jittering and/or alpha?
—	—	**Demo use-case 3**
3	—	Task: Find three test cases that reveal bug 8 & bug 1. Mention at the meeting (two most severe bugs so to make sure to bugfix these two faults we need three test cases from those faults)
3.1	3.1,3.2	Which of plots A and B do you prefer and why?
3.2	3.2	What information would you like to see in the label to help identify qualities that the test case might have? Ex. total number of mutants killed, coverage?

to solve the task and state the answers back to us, i.e., which tests they would select. We then asked them some questions related to working with the similarity maps. This was repeated for each set of visualisations. At the end of each set, we also asked about their preferred visualisation. The flow of the interview with the questions and different stages of the interview can be found in table 3.3.

4
Results

In this chapter we will describe initial results that we got from running our experiment. The chapter will be divided into sections and multiple subsection where we go into detail about the results we got for each research question.

4.1 Analysis

In order to analyse the gathered data and produced results, we will use different units of analysis. For RQ1 and its sub-questions (RQ1.1 and 1.2), we will be using average percentage fault detected (APFD) together with test case ranking, mantel test and visualisations. For RQ2, its sub-question (RQ2.1) and RQ3 we will be looking specifically at the visualisation. The visualisations will be evaluated differently depending on the research question

Average Percentage Fault Detected (APFD) is a way to calculate how fast a re-order test suit finds faults in the system under test [43]. In order to determine the APFD of a test suit T, we need to know the test cases it contains n, the number of faults that the test suit revealed m, and the position of the first test case that detects a fault TF_i.

$$APFD = 1 - \frac{\sum_{i=1}^{m} TF_i}{nm} + \frac{1}{2n} \tag{4.1}$$

This will help us choose the best distance measurement and test input-output combination based on the results as well as gain a deeper understanding of the impact of different distance measures and types of diversity in cost-effective testing.

Mantel Test is a test given two matrices that will produce a correlation score and a p-value. This will be done to check for correlations between the different distance functions. A correlation between the matrices can tell us how strongly or weakly each distance function is related to each other. We want to investigate this correlation by using varying data from the different repositories and as such observe how the correlation changes between the matrices.

Visualisations are going to be used throughout all 3 research questions. In RQ1 we will use the visualisations to plot the distance matrices produced by calculating the distances separately between test input and output. The types of visualisation produced will be histograms, box plots and heatmaps. In RQ2 we will use the visualisation in form of similarity maps the evaluate the effects that the dimensional

Table 4.1: Mean character count and standard deviation per file in each project. We measure the length for both test input and output.

Repository	Mean number of characters	Standard deviation
Grep input	45	6
Grep output	4416	25540
NanoXML input	396	347
NanoXML output	136	151
Sed input	93	58
Sed output	1094	1091

reduction has on the structure and composition of the said visualisation as well as the number, size and composition of clusters produced by different clustering techniques. In RQ3 we will ask our fellow students and software practitioners to give their critical evaluation of the produced visualisation aided by a set of question which will help us with the analysis.

4.2 RQ1 - Comparing Distance Measures and Types of Diversity

In this section, we will describe all the results which are connected to RQ1 and its sub-questions. RQ1.1 focuses specifically on the distance values that are contained in the distance matrices. These matrices are produced by measuring the distances between each test case with the use of different distance metric (Jaccard, NCD, Levenshtein) in a test suit. The measurements will be taken from both the input and the output files of the system under test. This will yield 6 different distance matrices (3 distance matrices x 2 test artefacts (input and output)). To evaluate these distance values will make use of ranking and the average percentage of faults detected (APFD).

In RQ1.2 we will take the resulting matrices and compare them as a whole i.e. we will be looking at the entire distance matrix as opposed to the values contained in that matrix. To evaluate the matrices we will use a mantel test that produces a correlation score and also we will be looking at the produced visualisation.
To better help understand the results from the following subsections, Table 4.1 shows how many characters are in each file. This is to better understand how different distance functions perform with varying lengths of the content it is reading.

4.2.1 Ranking and APFD

Using the MultiDistances.jl tool enables us to produce a prioritisation ranking for the test cases. By utilizing the data from the different repositories containing faults, we are able to calculate the Average Percentage of Faults Detected (APFD) and thus calculating which prioritisation order gives the best results. The results from

Table 4.2: APFD for all diversity types and diversity functions for all three repositories, Grep (440 tests, and 52 faults), NanoXML (216 tests, 8 faults) and Sed (336 tests, 4 faults). The highest APFD value for each project is highlighted in bold.

Artefact type	Distance function	APFD Grep	NanoXML	Sed
Input	Jaccard	**0.9409**	0.7330	0.7403
Input	NCD	0.9241	0.8380	0.7820
Input	Levenshtein	0.9236	**0.9302**	**0.9762**
Output	Jaccard	0.9104	0.8304	0.8432
Output	NCD	0.9239	0.8507	0.8811
Output	Levenshtein	0.9256	0.8547	0.8216
-	Random	0.9340	0.8575	0.8571

this procedure can be found in the table. 4.2.

Looking at the values presented in the table. 4.2, we can see that random produce results equal to the ordering produced by the diversity ranking. This is due to the test cases in each repository often revealing one to several faults per test case in the suit, making choosing a test case to execute next to trivial choice.

In figure. 4.1, we can see the percentage of faults detected for each 10th percentage step of tests executed. Again we see that the order created is quite redundant when random produce such a good result, where the curves look similar and there is no outlier.

4.2.1.1 Ranking results

In conclusion, the ranking created for the smaller repositories we have is quite redundant. Mainly due to the size of the software being quite small and that each test case has a high fault coverage in the repository which will lead to a higher APFD - no matter the ordering.

Grep has 52 seeded faults, of which 32 are found by test cases. This is quite a lot but due to each test case having a very high fault coverage, it would not really matter in which order you execute test cases, hence the high scoring and random performing so good.

Contrary, NanoXML and Sed do have lower amounts of seeded faults, 8 and 4 respectively and with a moderate amount of fault coverage per test. However, due to the number of tests and the low amount of seeded faults, it is very probable that a test case is able to find one of the seeded faults and thus the order again, has less impact.

The key takeaway here is that considering a smaller project with few faults or test cases with high fault coverage, the order of execution matters a lot less and random seems to generate an order that is as effective as an order based on diversity in many cases. However, with a smaller project where each test case has a lower fault coverage, it might prove effective.

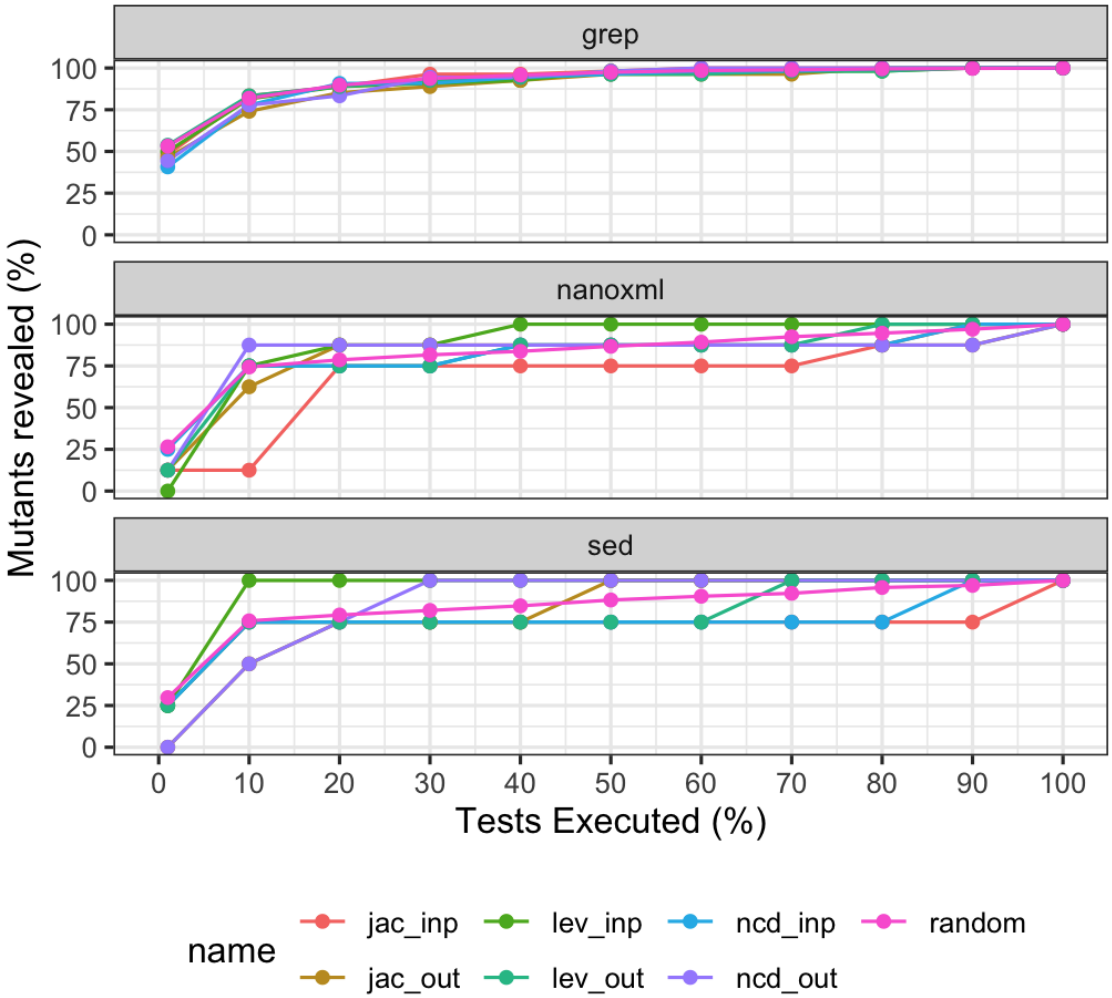

Figure 4.1: Amount of faults found after %X amount of tests executed, following order of APFD of varying diversity types and diversity functions

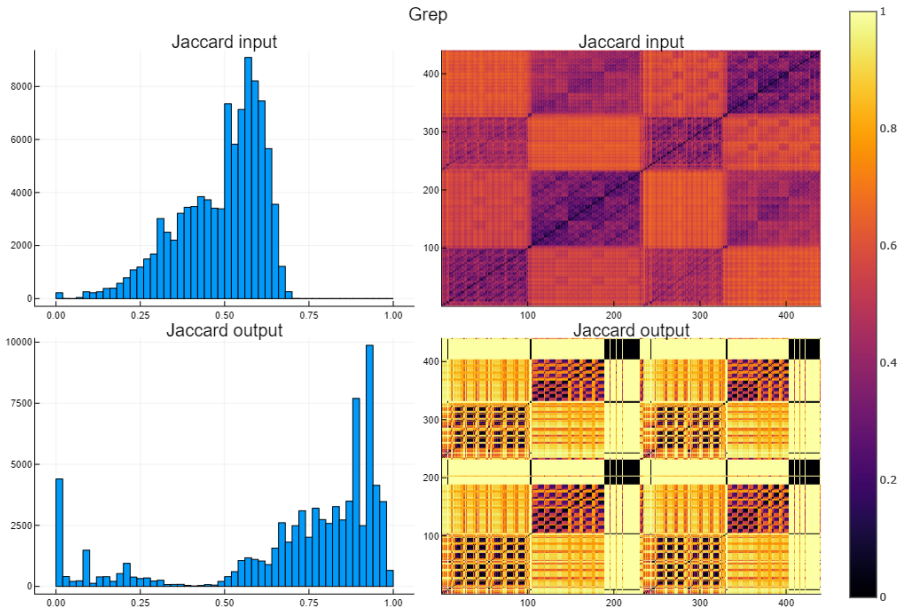

Figure 4.2: Resulting histograms and heatmaps from inputs in the Grep repository with the distance function Jaccard

4.2.2 Visual Analysis of Distance Matrices

The results from input and output performed by different distance functions are presented in the following subsections. The results are specifically connected to RQ1.1 where we are looking at the distance values in each distance matrix.

4.2.2.1 Grep

The following subsection talks about the distance matrices generated from the Grep repository's test input and output, visualised in the format of histograms and heatmaps.

In figure. 4.2 we can observe that the distance function Jaccard with input data (top row), produced quite a small distribution. This will be a theme in Grep for all visualisations due to Grep input data being quite small (30-50 bytes in the majority), hence the possibility of difference between the different inputs is lowered. In contrast, if we look at the output (bottom row), we can see a bigger distribution. Here the size of the output is generally bigger and more varied (40-358000 bytes), which in turn then gives us a bigger distribution. By observing we notice that there is a cluster of test cases that are identical, either by looking at the bar to the left in the histogram or by looking at the black spots in the heatmap. These are test cases that purposefully did not produce any output as a test.

Overall Jaccard produced quite an impressive amount of detail, considering it is the fastest distance function of the three distance functions we are testing.

In figure. 4.3 we can see that NCD produced a very small distribution compared to Jaccard. We do not get any finer detail here and everything is classified as very similar with no outliers. We can observe this in the histogram (top left), where we

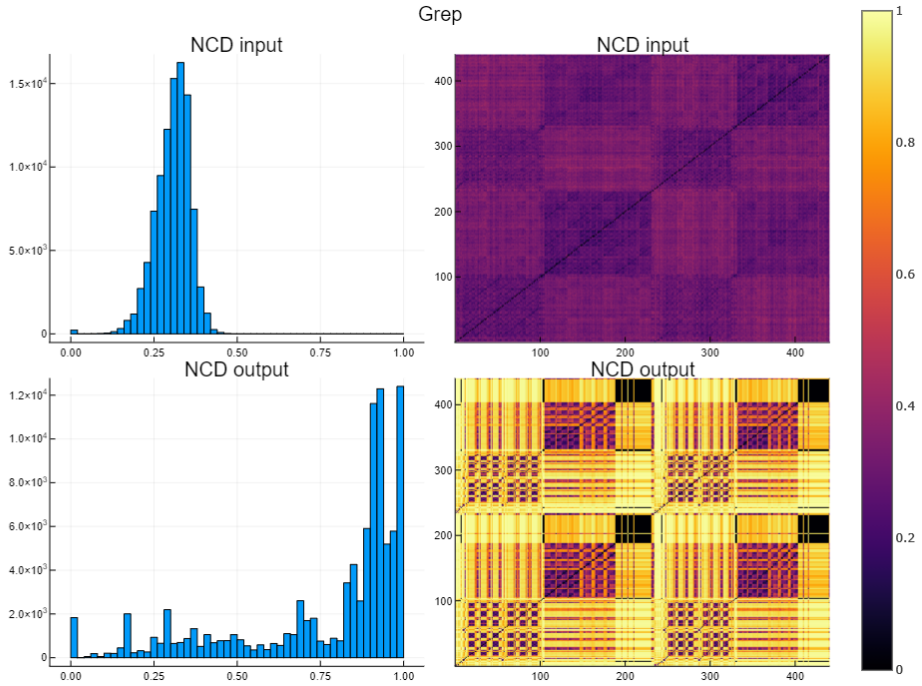

Figure 4.3: Resulting histograms and heatmaps from inputs in the Grep repository with the distance function NCD

see one lone peak, or in the heatmap where we can see dark purple colour with some small clusters that are more equal.

In the output, we can see a much bigger distribution, exactly as with Jaccard. The majority of the distribution in the output histogram is clearly leaning toward the right side of the plot, while the heatmap presents bigger areas with bright yellow. Compared to Jaccard we get more clusters of extreme values in the high end (right side of the histogram) and less towards the middle values, which indicates that NCD is less efficient with details compared to string distance functions (Jaccard, Levenshtein).

In figure. 4.4 we see the Levenshtein has produced a distribution quite similar to Jaccard but slightly more condensed, with a single peak that is more towards the right side, which would indicate the data overall being more similar with Levenshtein as the distance function. We can clearly see this difference if we compare the input heatmaps between Jaccard and Levenshtein, where Levenshtein has a darker hue, while Jaccard is lighter.

In the output, we see a very big spike on the right side, which indicates that we have many unique outputs. Compared to NCD and Jaccard we have many more unique results in Levenshtein and much more data towards the right side than any other place in the plot.

In figure. 4.5 we have a clear overview of the distribution and the mean and can clearly see in the input (left side of the plot) we have a broader distribution between Jaccard and Levenshtein compared to NCD. The mean of the data is higher in Jaccard than what it is in Levenshtein and NCD, which are close to each other,

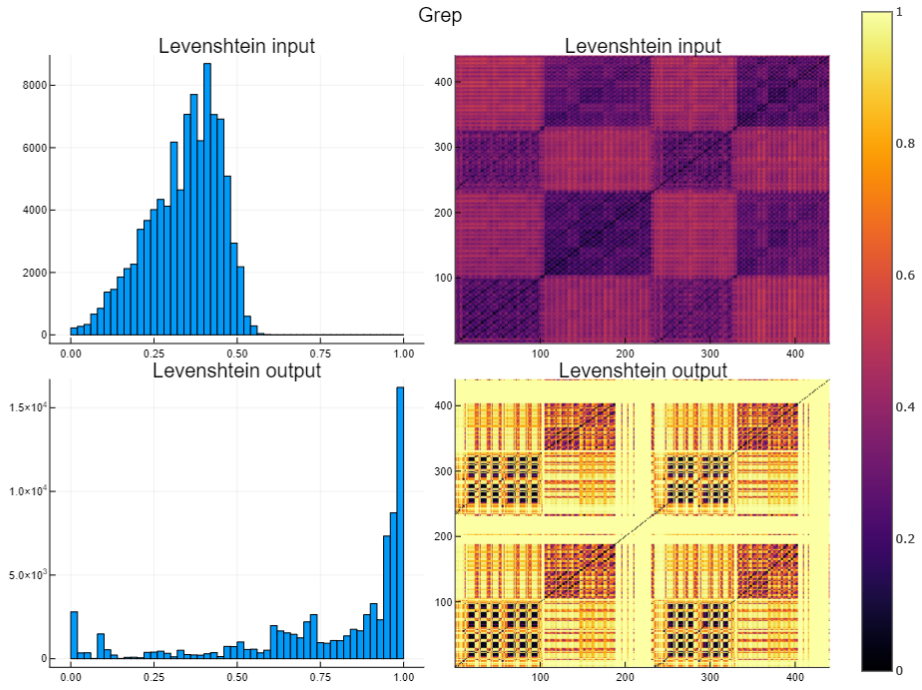

Figure 4.4: Resulting histograms and heatmaps from inputs in the Grep repository with the distance function Levenshtein

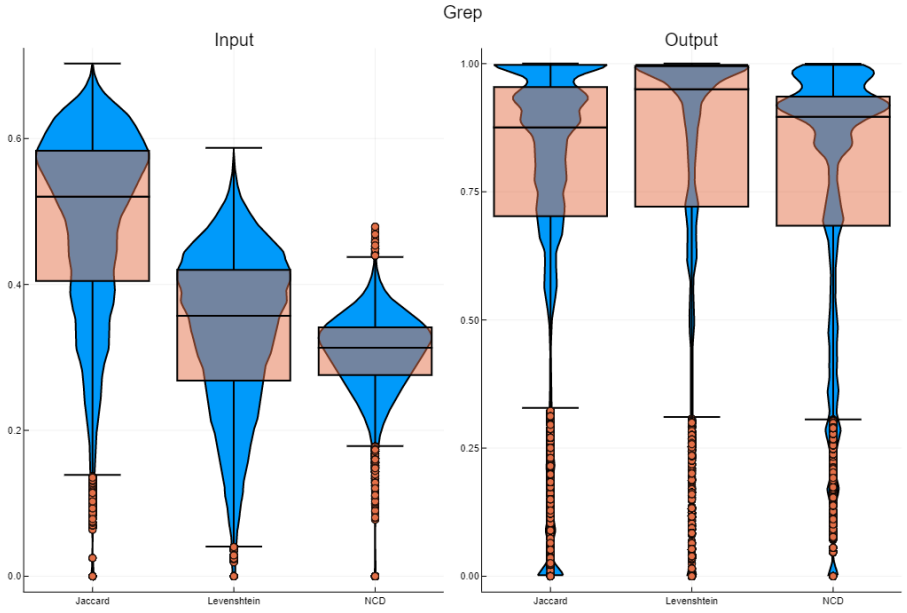

Figure 4.5: Resulting boxplot from inputs in the Grep repository with the all the distance functions under investigation

which means that Jaccard registered a higher disparity between the data.
In the output (right side), we can see that the distribution is fairly similar. Levenshtein having a higher mean, as we previously noticed in the figure. 4.4 where there is a big spike towards the high end, meaning there exist more unique outputs according to Levenshtein. Overall the different distance functions performed similarly in output where there is generally bigger data to inspect.

4.2.2.2 Results for Grep

In conclusion, we notice that NCD seems to have a more condensed distribution on inputs. This is due to the input being considerably smaller in size (text) and thus there is less to compress which yields a lesser difference. The string distance functions (Jaccard and Levenshtein) seem to capture the nuances of the text (as they should) due to being specifically for strings, as such they are able to capture different nuances in even smaller inputs, even though there might not exist many differences when there is a smaller input.
We specifically notice the differences in nuances if we compare the input heatmaps between each repository. We can see distinct clusters in all heatmaps, but in NCD specifically, it is much less distinguishable compared to Jaccard and Levenshtein.
In the output, we see that generally Jaccard, NCD and Levenshtein perform equally. The big difference between the input and the output is the size of the data. The output, as previously mentioned, is often times bigger, see one output size: 358000 bytes, compared to the lowly input sizes which are often around 30-50 bytes.

4.2.2.3 NanoXML

The following subsection talks about the distance matrices generated from the NanoXML repository's test input and output, visualised in the format of histograms and heatmaps. In Figure 4.6, we observe three big peaks in the input histogram (top left), that are positioned at each end of the spectrum of diversity and roughly in the middle. The peak to the left entails that we have many identical inputs according to the distance function Jaccard. We have two roughly equally big peaks located in the middle and far-right in the histogram, which means we have a pretty broad distribution, although skewed to the right. Comparing the input and output in this case, we can see that although we have quite a broad distribution of input, our output does not seem to be as broad.

If we observe input and outputs on the heatmaps we can observe the big area in the test space that seems to share many similar test cases which are not seen in the heatmap for output diversity. The reason is that input files are the same, but since the SUT performs different operations, the rendered output is different.

In Figure 4.7, we can see that what NCD produced is very different to what Jaccard produced. We do not have any identical input according to the input histogram, and if we inspect the input heatmap, we can see that the areas that are deep purple in the NCD (i.e, very similar) are black in the Jaccard input heatmap (i.e., identical tests). Note also that the data in the input histogram seems to be a lot more cen-

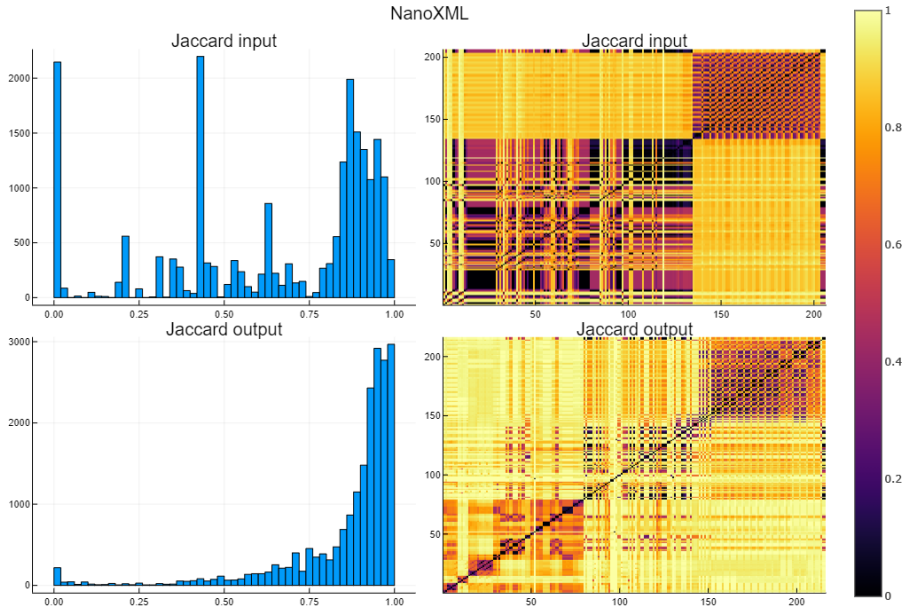

Figure 4.6: Resulting plots from running distance function Jaccard on inputs and outputs from the NanoXML repository

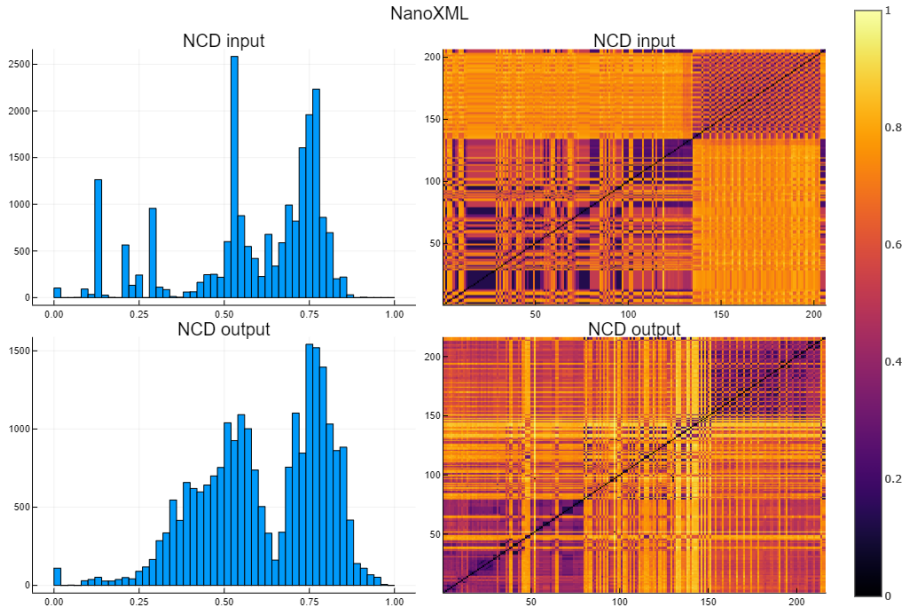

Figure 4.7: Resulting plots from running distance function NCD on inputs and outputs from the NanoXML repository

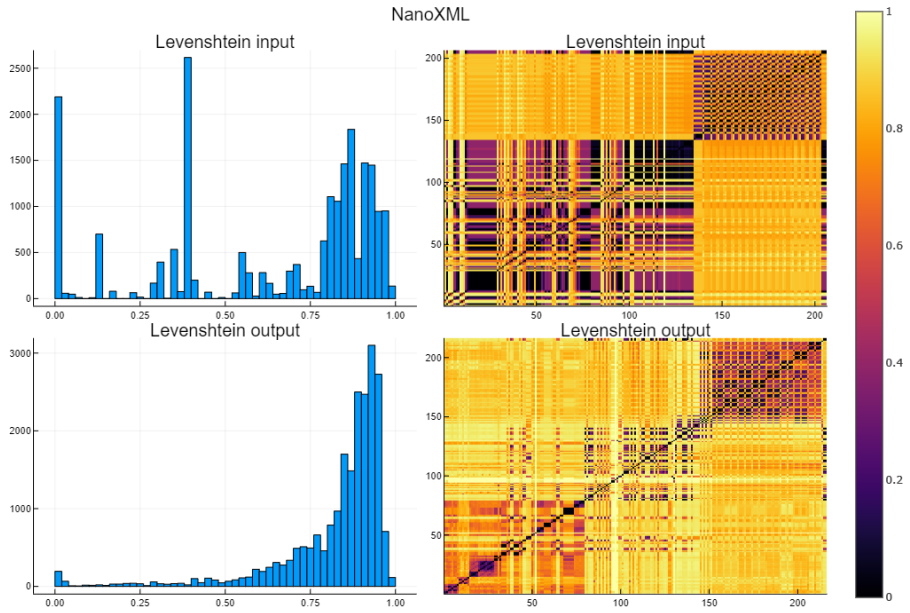

Figure 4.8: Resulting plots from running distance function Levenshtein on inputs and outputs from the NanoXML repository

trally focused when compared to Jaccard; in other words, for NCD none of the tests is identical or unique. The distribution overall seems to be broader in NanoXML for NCD compared to the Grep, which is due to the fact that the length of the input for NanoXML (file of circa 136 characters) is larger than the length of input used in Grep (strings with a mean length of 45 characters)—see Table 4.1.

In the output, we can see that the distribution is very different from Jaccard and Levenshtein as well. If we look at the histogram in the bottom left, we can see a more gradual increase towards the peaks in the data when compared to Jaccard and Levenshtein. However, if we look at the output heatmap (bottom right) we can see that structurally we have the same features between NCD, Jaccard and Levenshtein, with the hue being the difference mainly.

For Levenshtein (Figure 4.12), results show that both input and output are very similar to the Jaccard, both regarding the histograms and heatmaps. What is different, however, is that Levenshtein does seem to register less similarity in the higher end of the histogram when compared to Jaccard. Additionally, it also looks, as previously stated, very different to the NCD plots, specifically regarding the output.

Comparing the distance values in boxplots (Figure. 4.9), our results align with our detailed description of heatmaps and histograms. First, Jaccard and Levenshtein have similar distributions in input diversity, even though the test input itself is larger in length than the cases compared in Grep. Longer strings also seem to favour NCD, since the input diversity for NanoXML is higher when compared to the same measure but for the Grep project. Moreover, Jaccard and Levenshtein being string distance functions, still manages to capture the nuanced differences in the data better than

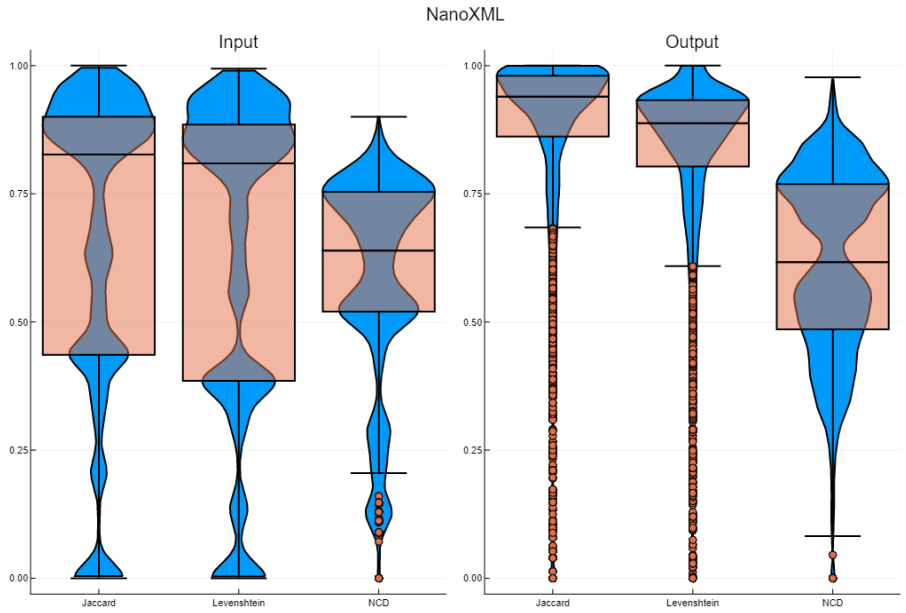

Figure 4.9: Resulting boxplot from inputs in the NanoXML repository with the all the distance functions under investigation

what NCD does. Lastly, the boxplots for output diversity are in contrast to the boxplots of output diversity in Grep. NCD has a broader distribution and fewer outliers, while Jaccard and Levenshtein has a more condensed distribution with many outliers, which we could see in the histograms previously as well.

4.2.2.4 NanoXML results

Looking at the input Jaccard and Levenshtein seem to have very similar results throughout all three different visualisations. They both have a wide distribution and similar medians, particularly, if compared to NCD. In other words, Jaccard and Levenshtein have very small differences between each other and NCD is the odd one out. Even though using NCD on input yields a moderate spread, the output distribution is also spread out and not converging into unique tests cases (as shown by Jaccard and Levenshtein in the output).

Similarly to Grep, the size of input/output files play a big part here. The input this time is bigger than the output producing a result for NCD which is closer to the other two distance measures 4.1 (mainly looking at the spread on both histograms and boxplots). The output result on the other hand behaves similarly to the results from Grep.

4.2.2.5 Sed

Figure 4.10 shows that the input histogram (top left) is skewed to the far right, which means that the distance function Jaccard deems most of the data to be diverse. Sed has larger input data generally (see Table 4.1), which makes it prone to

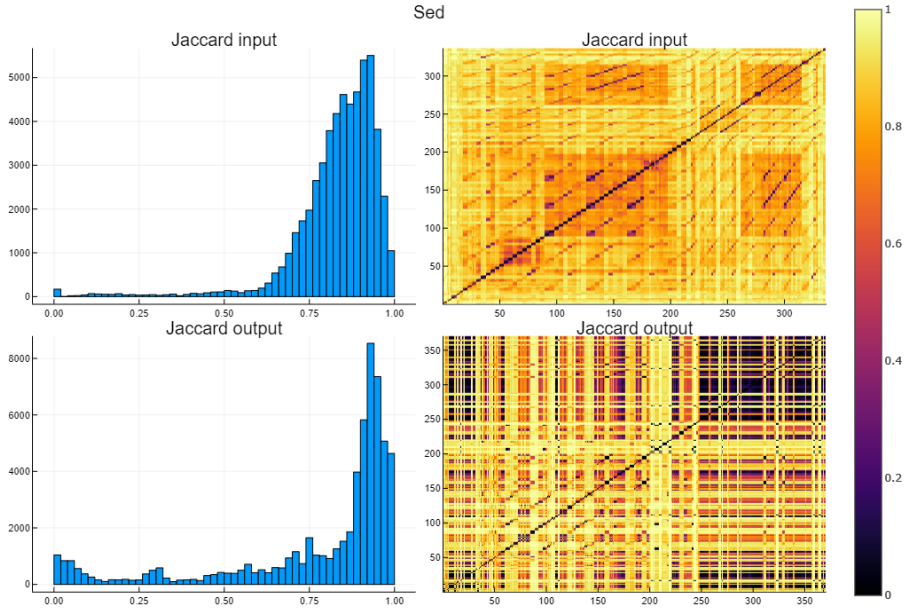

Figure 4.10: Resulting plots from running distance function Jaccard on inputs and outputs from the Sed repository

be more unique. We see the same effect if we look at the heatmap, where most of the areas have an orange/yellow hue. We see a similar pattern in the output diversity, except for more redundant tests (i.e., darker areas in the heatmap). That pattern is seen again when analysing results for NCD, except for having a distribution centered around 0.6 for input diversity (Figure 4.11)

For Levenshtein (Figure 4.12), we again see an input histogram very similar to what Jaccard produced, albeit the Jaccard input histogram is a bit more skewed to the right. In contrast, the histogram of output diversity (bottom right) has more identical tests than what Jaccard and NCD found. We do see the same areas in the heatmaps across all three distance functions identified similar areas, but Levenshtein seemed to have valued the areas generally higher than what the other distance functions did.

Lastly, the distribution in boxplots (Figure 4.13) show that we have very similar distributions in the input (left side) compared to the other repositories (i.e., $Levenshtein > Jaccard > NCD$), albeit on different places in the diversity spectrum. The mean differs quite a lot between Jaccard and NCD, while Levenshtein is slightly below Jaccard, despite some minimal overlap. For output diversity (right side), we have very similar distributions between the three distance functions, which we also saw at the Grep repository before.

4.2.2.6 Sed results

Between the three different distance functions, the results are generally very similar. In the input histograms, the general difference is that NCD is yet again more focused

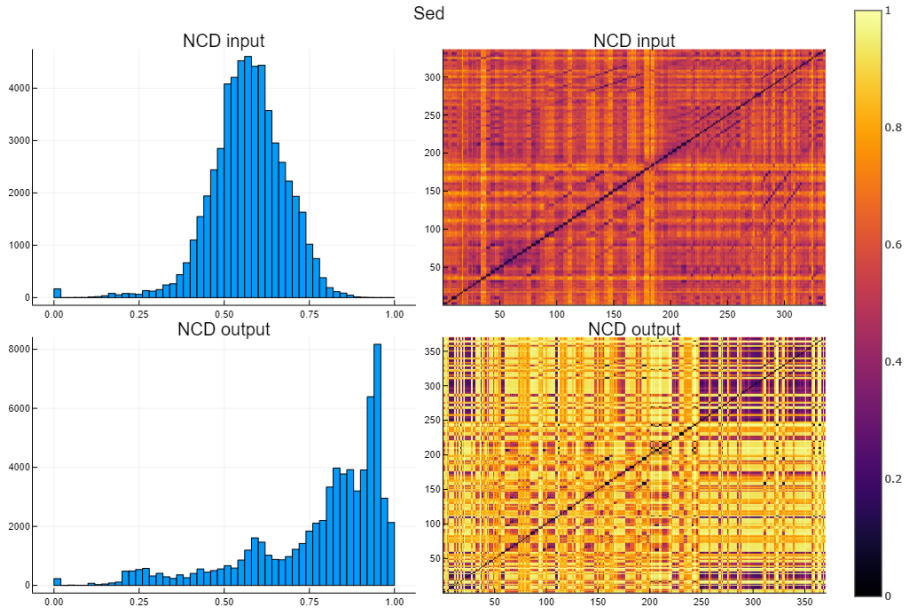

Figure 4.11: Resulting plots from running distance function NCD on inputs and outputs from the Sed repository

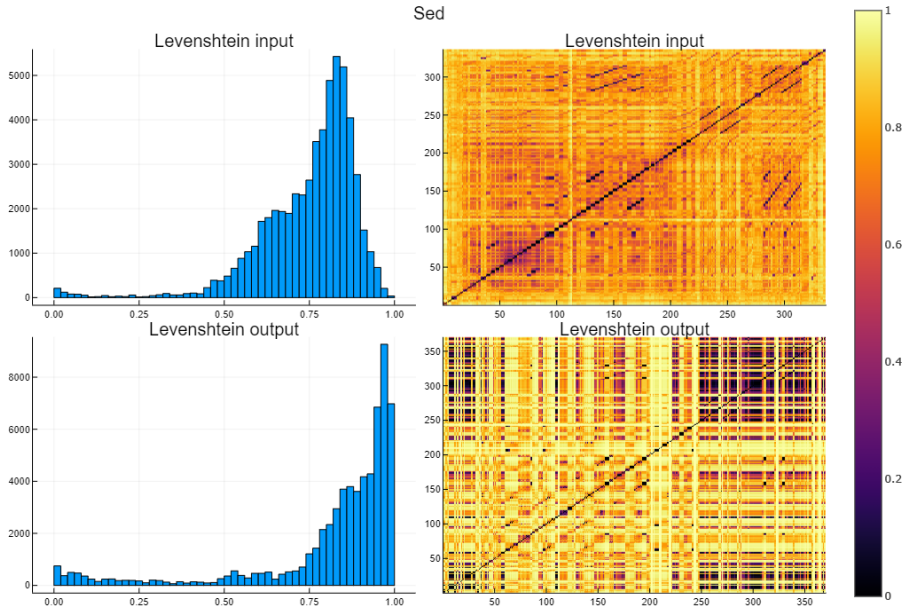

Figure 4.12: Resulting plots from running distance function Levenshtein on inputs and outputs from the Sed repository

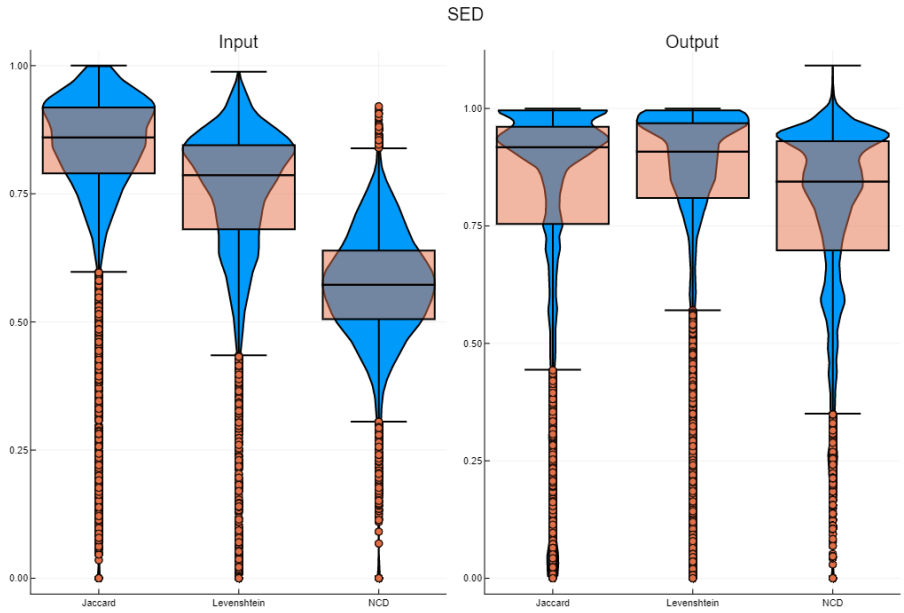

Figure 4.13: Resulting boxplot from inputs in the Sed repository with the all the distance functions under investigation

towards the middle of the plot, while Jaccard and Levenshtein register higher values. Again, this can be explained if we compare the size of the input, indicating that NCD generally produces less significant visualisations with smaller data. This is not surprising due to the trade-off of using compression in smaller strings. Meanwhile, the Jaccard and Levenshtein produce very similar distributions across the board, with minor differences of 0.1–0.2 in distance value which, we argue, is negligible in practice. All this can as well be seen in Figure 4.13, where Levenshtein and Jaccard are quite similar in terms of distribution and mean, while NCD differs.

In terms of output in the Sed repository, we see an improvement for NCD, which again, is due to the size for the output data being considerably larger which improves the benefits of compression that the NCD produces with text-based data. Jaccard and Levenshtein are once again also very similar to each other and if we look at the boxplots, all distributions from the three distance functions are very similar in terms of mean and distribution.

4.2.3 Testing Correlations between Types of Diversity.

In order to evaluate the distance matrices, we will use the Mantel test. The Mantel test will produce a correlation score and a p-value for each combination of distance metrics. This will be done for the test inputs and the test outputs giving us results for each repository, which will be presented in the table. 4.3. The method to compute the correlation between the distance matrices is the Pearson correlation coefficient and the method to calculate the statistical significance (p-value) is a two-tailed test.
By using the mantel test, we compare the correlation score between the different

Table 4.3: Correlation coefficient between the different diversity types distance matrices for the inputs. Highly correlated matrices (i.e., coefficient higher than 0.9) are highlighted in bold. The result for all items in the list are significant with a p-value of 0.01. For grep there are 440 elements, for NanoXML there are 206 and 216 elements and for Sed there are 336 and 370 elements.

Repo	Distance functions	Artefact type	Coeff
Grep	Jaccard-NCD	Input	0.74
Grep	Jaccard-Levenshtein	Input	**0.90**
Grep	NCD-Levenshtein	Input	0.76
Grep	Jaccard-NCD	Output	**0.91**
Grep	Jaccard-Levenshtein	Output	0.72
Grep	NCD-Levenshtein	Output	0.70
NanoXML	Jaccard-NCD	Input	**0.94**
NanoXML	Jaccard-Levenshtein	Input	**0.97**
NanoXML	NCD-Levenshtein	Input	**0.93**
NanoXML	Jaccard-NCD	Output	0.63
NanoXML	Jaccard-Levenshtein	Output	**0.91**
NanoXML	NCD-Levenshtein	Output	0.76
Sed	Jaccard-NCD	Input	0.47
Sed	Jaccard-Levenshtein	Input	0.79
Sed	NCD-Levenshtein	Input	0.51
Sed	Jaccard-NCD	Output	0.81
Sed	Jaccard-Levenshtein	Output	**0.91**
Sed	NCD-Levenshtein	Output	0.82

distance matrices. The goal is to assess quantitatively our findings from the visual analysis of heatmaps and boxplots. While the visual analysis can point to narrow patches of (dis-)similarities, the correlations revealed by the Mantel test can reveal how similar those distances matrices actually are to each other.

If we observe Table 4.3, we can see that in most cases Jaccard and Levenshtein are very correlated. In the previous subsection, we saw that Jaccard and Levenshtein were most of the times very similar with few disparities. We also see that, in many cases, NCD is far off Levenshtein and Jaccard, but sometimes correlates quite closely to Jaccard and Levenshtein.

The correlations also support the argument towards the influence of the size of the input on the differences between those measures. Grep had very **short input** strings such that NCD is not very correlated to Jaccard and Levenshtein. In contrast, the **longer output** strings in Grep revealed a higher correlation towards Jaccard. The same can be seen in NanoXML input, where the data is moderately large, we see that NCD is much more correlated than when operating on shorter strings. Looking at the output NanoXML and input of Sed which is considerably

smaller, we again see that NCD is not as correlated to Jaccard and Levenshtein.

In other terms, Jaccard and Levenshtein seem to be highly correlated when it comes to smaller data whereas NCD is not. On the other hand given large data all the distance measures are highly correlated i.e. there is not much difference between which distance function is chosen when operating with large data.

In conclusion, Levenshtein and Jaccard seem to be highly correlated in the majority of cases, with some variance when the data size is smaller, which makes Jaccard and Levenshtein a little less correlated, but still high. This means that Jaccard and Levenshtein basically produces highly correlated (with some variance) distance matrices relatively independent of size. By extension, we know that they produce similar data structures and thus perform similarly in the context of production.

4.2.4 Summary of RQ1 Results

Here we condense the results that were shown in the previous subsections by answering the research questions based on the data collected.

RQ1.1. How do the distance values differ when using various diversity measures and types of test artifacts?
The values produced by the different diversity functions seems to vary depending on the size of the input. NCD seems to suffer from this the most, which is mostly due to the compression on the smaller input, which leads to a lesser effect when compared to string distances such as Jaccard and Levenshtein which can capture nuances and details with smaller inputs. Jaccard and Levenshtein produce very similar values independent of the size of the data. Output diversity yielded higher distance values on all instances when compared to input values. However, those differences were also sensitive to the string length.

RQ1.2. How do the distance matrices differ when using various diversity measures and types of test artifacts?
Similarly to RQ.1.1. we see a big difference in the distribution produced by the distance matrices based on the size and composition of the data. NCD seems to produce very similar results to Jaccard when the data is bigger in size, whereas NCD is more sensitive to the string length and is less correlated to Jaccard and Levenshtein. In turn, Jaccard and Levenshtein often produce distance matrices that are strongly correlated with some variance introduced by the string length, where the size of the data lessens the correlation. The differences between input and output artefacts also seem negligible, where the difference between the composition of the distance matrix relies, again, on the length of strings.

RQ1. What are the trade offs in diversity for using different diversity measures and types of test artifacts?
The big trade-off between different diversity measures is that both string distances used in this study tend to fare better than the compression distance on

Table 4.4: Shapiro-wilk test results

Diversity function	**p-value**
Jaccard	0.48
NCD	0.57
Levenshtein	0.46
Artefact type	**p-value**
Input	0.10
Output	0.22

data that is smaller in size. Both of the string distances (Jaccard and Levenshtein) have similar results across the repositories involved in this study and can capture more nuances in the data than the compression distance, this is specifically evident in smaller data and not as much in larger data.
Another factor to keep in mind is run-time for each distance function, where Jaccard outperforms both Levenshtein and NCD on all types of data. NCD performs faster than Levenshtein on bigger data, while Levenshtein outperforms NCD on smaller data.

4.2.5 Statistical analysis of APFD and Ranking

Here, we verify our null hypothesis: *There is no difference between the different artefact types and diversity functions in terms of APFD score.*

In order to perform a statistical test on our data we first need to check if our data is normally distributed. Checking for normality of data will allow us to decided between parametric or non-parametric test i.e. if the data is normally distributed we need to do a parametric test and if not a non-parametric test. We use the Shapiro-Wilk test to check for the normality of our data, we chose this specific test because we are the most familiar with it. The results from the test can be seen in the table 4.4. Here we can see that none of the p-values is lower than the chosen alpha of $\alpha = 0.05$, which means that we cannot reject the null hypothesis that the data **is** normally distributed. Although we cannot reject the null hypothesis we still went for a non-parametric test because our data was quite small and is very unlikely to be normally distributed.

The results from Kruskal-Wallis statistical test can be seen in the table 4.5. Here we compare the APFD scores between each diversity function and artefact type.

Table 4.5: Kruskal-wallis test results

Diversity functions compared	**p-value**
Jaccard-NCD	0.42
Jaccard-Levenshtein	0.14
NCD-Levenshtein	0.20
Artefact type compared	**p-value**
Input-Output	0.75

With a chosen alpha of $\alpha = 0.05$, our results indicate that none of the diversity functions have statistically significant different results since the $p - value < \alpha$. This means that we cannot reject the null hypothesis. The APFD scoring between the different diversity functions and the artefact types, do not produce a scoring that is significantly different from one another.

Table 4.6: Hyperparameters for t-SNE, UMAP and DBSCAN. The cells with dashes indicate that this particular parameter is not applicable to the corresponding dimensionality reduction or clustering technique.

Repoistory	Hyperparameter	t-SNE	UMAP	DBSCAN
Grep	No.Iterations	3000	-	-
	Perplexity	30	-	-
	n_neighbours	-	30	-
	min_dist	-	0.125	-
	epsilon	-	-	0.20
	minPts	-	-	10
NanoXML	No.Iterations	350	-	-
	Perplexity	14.35	-	-
	n_neighbours	-	10	-
	min_dist	-	0.7	-
	epsilon	-	-	0.15
	minPts	-	-	10
Sed	No.Iterations	5000	-	-
	Perplexity	18.35	-	-
	n_neighbours	-	20	-
	min_dist	-	0.05	-
	epsilon	-	-	0.5
	minPts	-	-	10

4.3 RQ2 - Comparing Dimensionality Reduction Techniques

Here, we compare the degree of impact that each dimensionality reduction technique and our chosen clustering technique have on the similarity maps. We apply dimensionality reduction and the clustering technique on the **input** distance matrices generated using Jaccard in all cases. Based on RQ1, we choose Jaccard because it performed similarly to Levenshtein and outperformed NCD. We first use the clustering technique on the raw distance matrix that we got from RQ1 and then proceed to reduce the dimensionality of it with the dimensionality reduction techniques.

When it comes to dimensionality reduction techniques such as t-SNE and UMAP, some hyperparameters have to be chosen in order to fine-tune each technique for each distance matrix. The same had to be done for DBSCAN to determine the number of clusters. In order to determine the hyperparameters, we closely followed the recommendations given in the papers corresponding to each dimensionality reduction technique and clustering technique. Furthermore, we applied our domain knowledge and performed visual analysis to further help us deciding on the hyperparameters. The full list of hyperparameters can be found in Table 4.6.
The results from the dimensionality reduction are presented in the following sections.

For each DBSCAN visualisation, we decided to remove the outliers, which are data points that were not fitted into a cluster. Since there was not much difference between each dimensional reduction technique, we chose to omit the results from the Grep repository and put them in the Appendix. However, we still think it is important to not remove those results entirely as they still bring some insight i.e. for this specific repository no matter which dimensionality reduction technique is used the result will be the same. The remaining projects (NanoXML and Sed) show more interesting results and variance, hence leading to more in-depth discussion.

4.3.1 NanoXML

The following section is the resulting visualisations from running dimensionality reduction techniques with DBSCAN. For each subsection, we will go through different dimensionality reduction techniques and comment on the results.
NanoXML - MDS: In Figure 4.14, we can see 6 distinct clusters, for which the colours are displayed in the legend. We decided as mentioned at the beginning of this chapter to turn off the outliers (value 0 in the legend) for clarity. There are two clusters that visually are shown as a single point in the plot. However, that is misleading because, in reality, there are at least 10 data points hiding under one since they overlap. We can further prove that since we have tuned DBSCAN to make a cluster if there are at least 10 points (see table 4.6). We decided that 10 data points within a certain diameter (ϵ) should constitute a cluster using our domain and data knowledge regarding the repository.

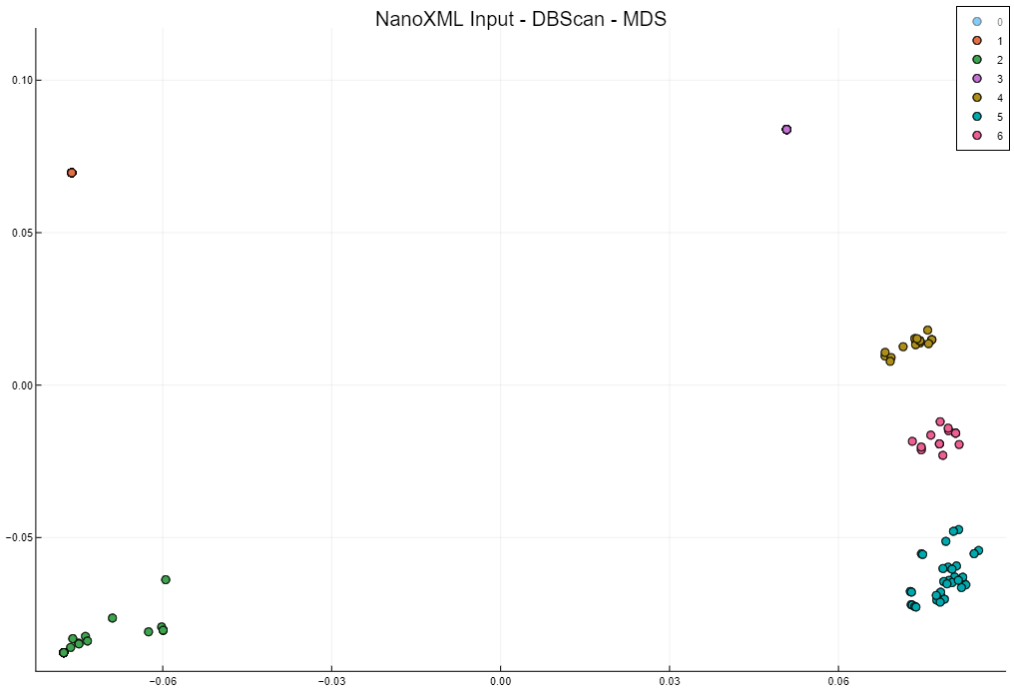

Figure 4.14: Dimensionality reduction MDS on the input data of NanoXML repository using DBSCAN clustering

NanoXML - t-SNE: In Figure 4.15, two spherical like clusters are immediately brought to our attention. The clusters form like that because there is a lot of data points that "overlap". This can also be explained by looking more closely at how t-SNE puts points on the graph. Each point has a minimal distance to another pushing the points out from the centre forming a ring-like cluster. This structure is only prevalent when there is a lot of points on top of each other. This characteristic can be observed in clusters 1, 2 and 3 in Figure 4.15.

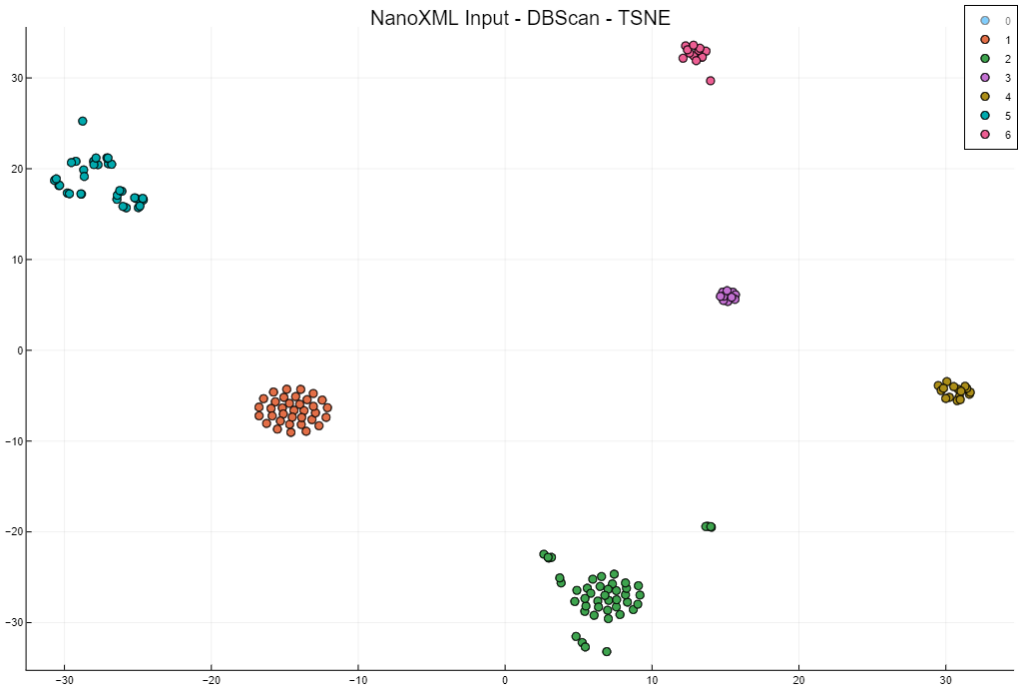

Figure 4.15: Dimensionality reduction t-SNE on the input data of NanoXML repository using DBSCAN clustering

NanoXML - UMAP: In comparison to MDS, UMAP, similarly to t-SNE, captures and shows the points which are layered on top of each other as evident in figure 4.16 when we look at clusters 1,2 and 3. The way they are displayed is a bit more natural without any specific shapes forming as opposed to in the t-SNE graph 4.15. There could be a point made that t-SNE makes a better case when it comes to anomalies such as a lot of points having the same x,y coordinates.

4.3.2 Sed

The following sections show results in form of visualisations from running different dimensionality reduction techniques on the distance matrices produced from the inputs to the Sed program.

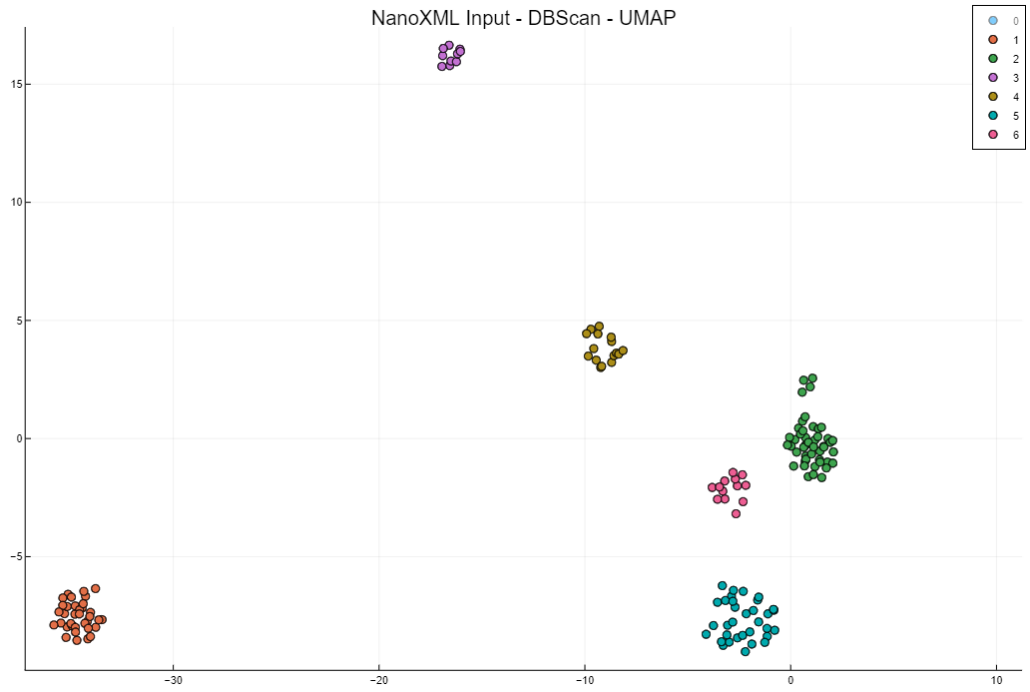

Figure 4.16: Dimensionality reduction UMAP on the input data of NanoXML repository using DBSCAN clustering

4.3.2.1 Sed - MDS

Figure 4.17 shows a graph that has hardly any shape to it. The data points are spread out everywhere. The same can be said about the clusters which are not really positioned closely. This specific repository also had a lot of outliers as can be observed by looking at the box plots 4.13 (subsection 4.2.2 in RQ1) from our previous research question. This could be the potential reason why the data points and the clusters are spread around like that.

4.3.2.2 Sed - t-SNE

In comparison to MDS, t-SNE adds more of a structure to the overall layout of the graph 4.18. Here we can easily identify clusters and the data points are not as spread around the graph but are more tightly coupled to the cluster they belong to. On the other hand, the clusters themselves are spread out. The graph also has 3 anomalies, namely, there are three points that are far away from the clusters they belong to and closer to other clusters. This could potentially be explained by DBSCAN not doing well and miss-assigning points when the data has a large spread and there are many outliers present.

4.3.2.3 Sed - UMAP

Figure 4.19 shows the result from UMAP. Here we see the same situation as with t-SNE, although the clusters are not as tightly coupled.

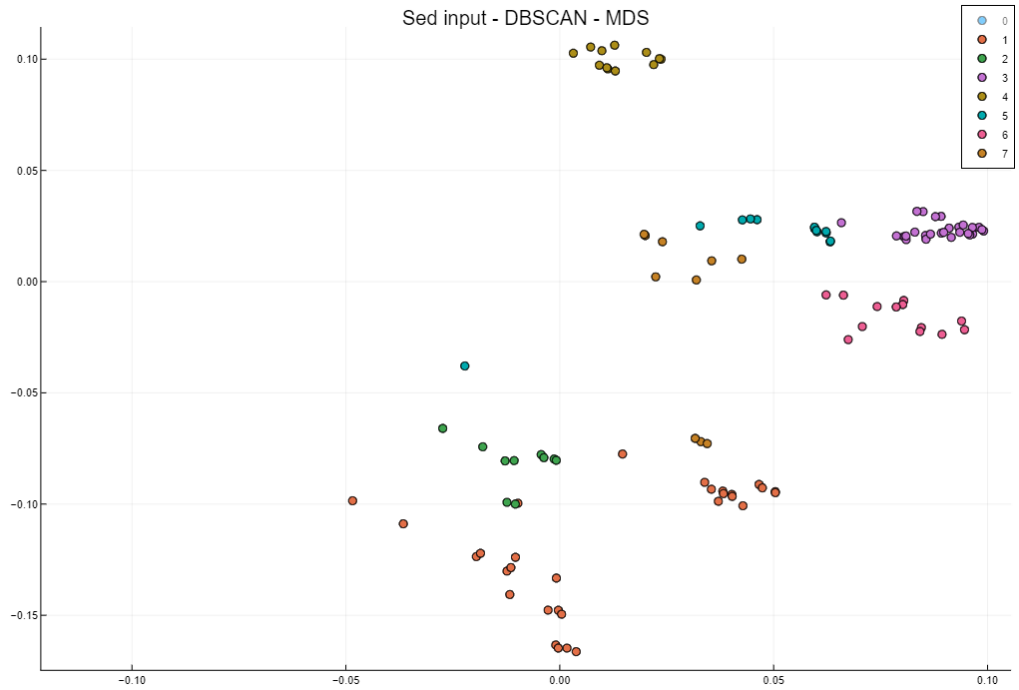

Figure 4.17: Dimensionality reduction MDS on the input data of Sed repository using DBSCAN clustering

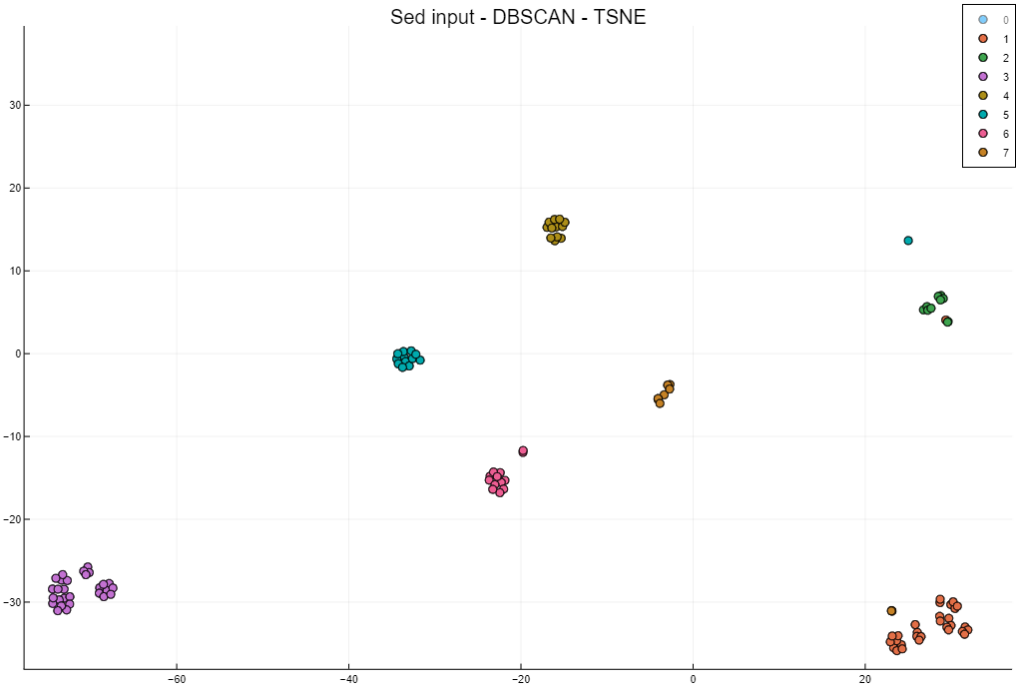

Figure 4.18: Dimensionality reduction t-SNE on the input data of Sed repository using DBSCAN clustering

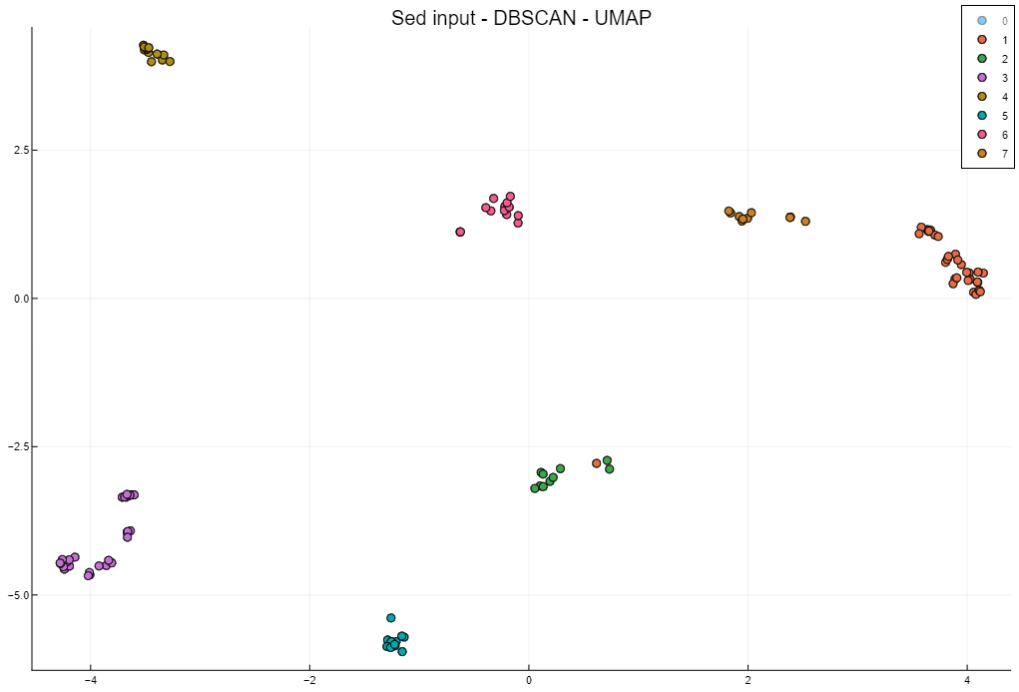

Figure 4.19: Dimensionality reduction UMAP on the input data of Sed repository using DBSCAN clustering

RQ2 What are the visual and structural differences on the clusters after using different dimensionality reductions?
Depending on the data which is fed into the dimensionality reduction functions the results may vary from being the same across all techniques to differing slightly. Generally, t-SNE and UMAP fair better on data with more spread, grouping the points from the same clusters closely and spreading the clusters around the graphs while MDS does not have much of an effect on the overall layout of the data displayed on the graph. The second point which is worth mentioning is that t-SNE and UMAP visualise points with the same x,y coordinates better than MDS, giving the user of those functions much more depth to the data.

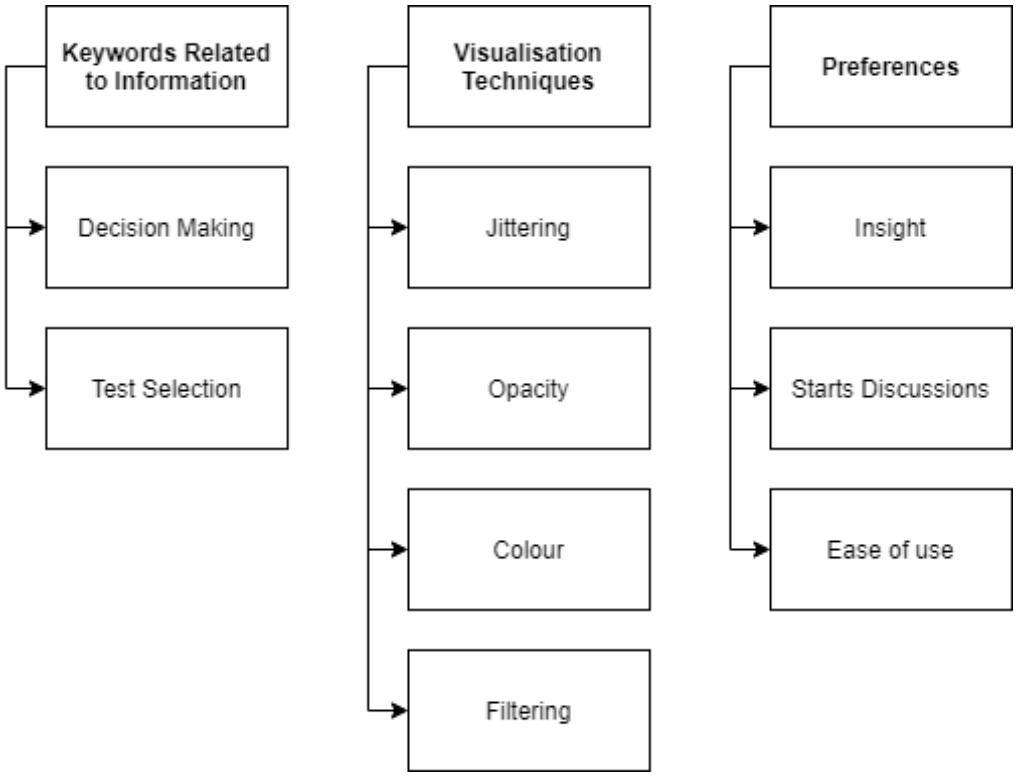

Figure 4.20: Core themes and their sub-themes

4.4 RQ3 - Participants' Preferences on Test Visualisation

In this section, we will write about our findings and results from doing the interviews which are connected to RQ3 and its sub-questions.

Interview analysis: In order to analyse the data gathered from the interviews we performed qualitative analysis and theme analysis. We choose relevant segments of text (codes) from the interview transcripts and aggregated them under their respective code labels. We then collected all the code labels and summed them up under a theme.
In order to be more straightforward in our coding process, we did selective coding [44] in which the main categories were defined prior to the start of the coding. The sorting of codes into themes and sub-themes then revealed sub-themes that were then mapped to different aspects of our RQ3 (i.e., visualisation techniques, preferences and solving the assigned tasks). The codes revealed dedicated discussions on various aspects of the visualisation (e.g., opacity, colouring) and the participant's preferences. An overview of themes and sub-themes is presented in Figure 4.20.

4.4.1 Test Selection

As outlined in our methodology, we gave the interview participants a number of different visualisation along with tasks that involved selecting test cases. The thought

process behind test case selection of each participant was quite similar: begin by looking for a cluster of test cases regardless of the visualisation shown and, then, sample tests from different clusters.

> "*Yeah. I would say the same reasoning. I'm not sure if I pick the same test cases as I did before, but I used the same logic as before, that looking at the clusters and then, trying to see the extreme values*" - Person 5

The participants had differentiating opinions after choosing test cases from their first cluster. Some looked at clusters that were more distant from their first choice others utilised tooltip information of each data point to take their next pick. The tooltip included information about the name of the test.

> "*...Similarly here, obviously I'm going to go look at like clusters, but yeah, you would probably pick one, which has like five faults, right?..*" - Person 4

A point of interest and what also influenced the test selection process for some participants was the different visualisation techniques that we have implemented for the presented visualisations.

> "*...I would completely ignore the right area of the screen, right? So I would completely ignore that part because, nothing really stands out to me, right? So what I would probably do is to go with one of those because they appear to be slightly brighter or because I'm seeing some red coming through here...* " - Person 3

One of our visualisations contained a gradient colour that went from black to brighter hues of red as the test cases found more faults. The person from the above quote chose to ignore a large cluster of test cases that were darker even though some of them did in fact find faults. When presented with another visualisation that had discrete colours per fault (i.e., if a test case found 1 fault the colour would be for example blue if it found 2 faults it would be green, etc.) the participant chose differently, by selecting a test case from the previously ignored area.

In contrast, other participants thought that different visualisation techniques don't bring too much to the table when it comes to choosing tests and see them as just different means to get to the result. The specific test cases that the participants have chosen were not the same, i.e., the names of the test cases (IDs) were not the same. After each visualisation, the participants often chose different test cases but many of them came from the same cluster. The only time that the participants have chosen the exact same test case was during task 3. This was a special case where a specific bug was only covered by one test case.

Although the tactic does not seem to change between different visualisations, i.e., choosing a test case from each defined cluster, another dimension added to the visualisation such as colour or size of data point, often influences the person to perform a more calculated decision to, e.g., pick a test case that reveals more faults or a specific fault. In contrast, when only shown a regular two-dimensional visualisation without colouring and size, they often times chose tests that did in fact not even reveal a fault in the repository.

4.4.2 Jittering and Opacity

In our investigation of the preferred visualisation techniques to visualise diversity, we have found that jittering and opacity are helpful to some of our participants. Jittering is a technique that pushes two data points apart when they have the same coordinates, whereas opacity is the transparency of a data point in the plot.

> "*I think opacity is helpful, right? Uh, because. Again here, right? Looking at this cluster, right? We can see that there is actually a lot of test cases here, before it was a single dot so it was very difficult to, to, to understand what was going on, Right? But looking at it here now we can see that it is a pretty dense cluster with a lot of, with a lot of test cases, uh, and you know, there is a number of them which are finding, uh, you know, quite a big number of issues. Right. So, uh, yeah, opacity, having the jittering on is definitely helpful.*" - Person 3

One participant thought that the jittering and opacity options added no benefit but as the interview progressed and the participant was faced with different tasks, the participant began to change their mind.

At the start of the interview:

> " *I don't really see the point of jitters. I don't really see the point there if you want to choose the, the test cases. I can't, I can't see the benefit though.*" - Person 1

As the interview progressed:

> "*I would prefer that no, opacity..., Hmm,no. I take that back... In this case, because when you're in the default view... Then you can see, they are very close together in that way. Do you understand what I mean by that?... Yeah. So that's, that's it. That can be useful. So I take that back. I think you can use that for this kinds of graphs* "
> - Person 1

Finally, some of our participants liked the jittering and opacity to be enabled in some visualisations but disabled in others. The reason is that they found those options confusing for some of the tasks that they were given.

> "*I think now I'd prefer to have the opacity/jitter on, Because I think I can, clearly kind of differentiate, now that it's just black and red, so, it kind of differentiates, which part of the cluster, I can see more reds and I can choose a test case from*" - Person 5

> "*They're stacked on top of each other more than the fact that I might have. Eh, designed bad tests because they are the same input is the same. Um, but if I'm just choosing random tests, then the information is not needed.*" - Person 4

4.4.3 Colours

Most of the participants liked to work with visualisations that had discrete colours as opposed to visualisation with a colour gradient.

> "*...having a discreet color between the passing tests and failing tests is really good as well...*" - Person 3

> "*I thought it was easier for me to see, to distinct them. Like the sizes and the red against the black. Also I have to announce that I have a form of colorblindness.*" - Person 1

The colour palette that was used on the visualisation with discrete colours was quite random and didn't follow any previous pattern, for example when the test case found 1 fault the data point corresponding to it would be blue, if the test case found 2 faults, it would have a pinkish colour. Generally, there were no bad comments about the colour palette even when one of our participants suffers from colour blindness. Only one participant had an issue with the colour palette and would like it to be more natural or, as the person puts it, more emotionally aligned. In other words, green being good meaning no faults found and red being bad meaning that faults are found.

> "*if the colors were more, you know, uh, as I said kind of emotionally aligned, I would like that one as well. (referring to Figure A.6)*" - Person 3

None of our participants gave positive feedback when it came to working with the visualisation that had a gradient colour scale. Some participants didn't care much about it but, for those who did, it felt like the colour gradient gave off misleading information. This might be because test cases that revealed only *one* bug had almost the same colour as faults that revealed *no* bugs. As mentioned previously, during the test selection one participant chose to ignore a section of a graph just because of how the colour gradient was set up.

The most positive feedback we received was on the visualisation with discrete colours: red and black (red meaning a fault was found and black meaning there were no faults found). This graph (Figure A.8) had an additional visualisation effect which might have given it an edge over the other graphs in the set. The visualisation effect in question, instead of a different colour for the number of faults found, we increased the size of the data point, i.e., the more faults a test case found the more it grew in size in conjunction with binary colours.

> "*I think the final one (Referring to Figure A.8), which was more, clear with the red and black cluster. And then they're just like two differentiation. It's either red or all black*" - Person 5

4.4.4 Filtering

Filtering techniques were used in the first and third sets of visualisation. The first set included a filter of ranks which was implemented as a slider. The slider could be moved from left to right and vice versa in order to display the data points on the graph. While moving the slider the test cases showed up or disappeared one by one, i.e., the slider moved in increments/decrements of one. In our investigation, we asked the participants if they would prefer the slider to be percentage-based. In other words, the data points would show up in percentage increments. For instance, when the slider would move up every 10%, more test cases would become visible. We found contrasting opinions about that feature. Some participants liked the idea

to use percentages and thought that this would be more intuitive for less technically inclined people, e.g., managers or employees that do not know the lower-level details of the SUT. Whereas, as a developer, they would not be affected by that specific feature. Interestingly, many of our interview participants did not really use the filtering option. Most would just look at the overall picture of the visualisation and then select tests, instead of using the filtering to verify, e.g., what are the 10% most diverse test cases.

> "*To be honest with you, I don't have a strong opinion on it. Percentage might be more generic, uh, and. It might make more sense to people that are not as technical. As a developer, it's honestly, doesn't make a big difference to me, uh, whatever it is, right*" - Person 3

The next set of visualisations that contained a type of filtering were for task 3 (filtering based on fault coverage). One visualisation contained a slider whereas the other contained categories (via a box) that could be clicked to be enabled/disabled from the plot. Both filtered the visualisation based on the fault found i.e. each slider step was a different fault, so when scrolling from left to right we only displayed test cases that covered one fault at a time. The visualisation with a legend worked in a similar way but instead of showing test cases that only covered one specific fault, the box could enable/disable many faults at once. The participants could press on the legend to isolate a specific fault on the visualisation or leave only a set disabled. Most of our participants preferred the visualisation which contained the interactive legend box.

> "*...it is an easy representation. You can have everything in the same graph, so you can see it a lot easy on the whole pictures.*" - Person 2

Tying back to filtering options not being utilized to a higher degree, it is worth mentioning that given enough time (for example a couple of hours) a learning effect will come into play. The participants would learn how to use the visualisations and hence their perception towards filtering techniques such as the slider might change and the value of the filtering options might increase or decrease.

RQ3.1. Do different visualisations motivate testers to select similar/different sets of test cases for the same tasks?
Different visualisations do in fact motivate the testers to choose different sets of test cases. Particularly for the graphs that use colouring as a means to convey information such as faults. In addition, discrete scales tend to help the tester with the comprehensibility of such information.

RQ3.2 What types of filtering, colouring and visual attributes are preferred when visualising diversity?
Generally, the participants did not interact much with the sliders but welcomed it as more of a tool that could spark discussion between the developers and less technically inclined people in the company. Regarding colours, all interview participants preferred scales to be discrete. Generally, fewer colours are preferred.

Even better if the colouring is connected with a sort of visual attribute such as the size of shapes indicating how many faults are detected.

RQ3 To what extent do different visual techniques influence decisions about test case selection?
In our investigation different visual techniques influence the decisions about test cases selection at a moderate rate, although given more time this rate could be subject to change due to maturation. Some visual techniques do not bring much insight or use to the tester such as sliders, whereas others have a great impact on the decisions making such as different ways of using colours to present the data. In general, interview participants liked seeing an overview of the entire test suit first before going into more detailed tasks and decisions while using other filtering techniques. This is where the use of colour shines the most since we can see interesting sightings from the get-go.

5
Discussion

In this chapter, we will further discuss the practical impact of our results referring back to the research questions answers that we gave in the previous chapter. Additionally, we discuss our threats to validity and future work.

5.1 Trade-offs in using different diversity measures and types of test artifacts

In this study, we examined the different trade-offs in using varying distance functions and test artefacts, specifically the input and output of test cases. In our study, we handle textual artefacts, which means that the input and output are in the form of text. This is an important distinction because NCD is able to compare any type of information and is independent of the application. In contrast, Jaccard and Levenshtein are specific to textual artefacts and, as such, are dependent on the application under test.

5.1.1 What distance function to use?

As depicted in the results section, we see that NCD does not manage to capture small nuances in smaller strings, while Jaccard and Levenshtein were able to discern the smaller details of strings with varied sizes. NCD itself does perform better on bigger data as opposed to smaller data, where the compression of the data is more impactful. Jaccard and Levenshtein however, process the text itself without compressing and as such is able to capture details, such as tokens in the data, or in the case of Levenshtein sentences or names, even if the string is short.

Generally, Jaccard and Levenshtein managed to capture most details, while NCD was expectantly lagging behind a bit and as such the advice with textual artefacts would be to go with either Jaccard or Levenshtein. However, it is worth noting that Levenshtein is very slow in processing larger data, compared to Jaccard and NCD. This means that if there is a big project, the suggestion would be to go with preferably Jaccard, while with smaller projects Jaccard or Levenshtein both performs equally good, so the effect of the choice is negligible.

5.1.2 Input vs output artefacts

In the study we performed, the only results we could find was that the output distribution generally seemed to produce more unique data compared to input. A reason for this as we can see in Table 4.1 is that generally, the output had a higher mean string length, which means that the data was larger. This in turns leads to more unique data in most cases.

Generally, when choosing between input or output diversity, we recommend going with the option that yields the largest data pool to benefit from a more unique set of diversity data. This decision has to properly balance time and resources since very big data will be very slow to analyse. So going with too big data might be a huge downside. The best recommendation we can give is to first and foremost use domain knowledge on which data is more suitable, but bigger data seems to be a better choice compared to smaller data when there is a lack of data knowledge.

5.1.3 Additional evaluation metrics for diversity

In this paper, we used APFD to show the effectiveness of diversity. However, this is not the only metric that we can use to show this effectiveness. An alternative metric that adds value to the evaluation is code coverage. In our case, the inclusion of such metric requires additional manipulation of the source code and additional time which we did not possess and as such is a delimitation of our research.

Another metric that can be used is test execution time to measure the effectiveness of diversity. This would tell us the time gained after prioritising the test suit. However, one thing to keep in mind is that most tests had the same test execution time (roughly 100's of milliseconds) and thus the primary gain would be to simply reduce the test suit overall. This combined with APFD or coverage would add value to this thesis, since time to execute is a big factor in many industrial applications, such as real-time systems.

5.2 Visualising test diversity data with dimensionality reduction techniques

Depending on the situation there are different answers to the question, "What dimensionality reduction technique to use?" If one does not have domain knowledge or little knowledge of the data, we would recommend using MDS. The reason is, its simplicity to configure, such that it does not require complex parameters or tweaking. On the other hand, a strong disadvantage that we saw was that MDS did not capture data groups in tighter clusters as t-SNE and UMAP did.

In contrast, for testers more familiar with clustering or more familiar with the data being used (test input/output), we recommend between t-SNE and UMAP where both techniques perform similarly to each other. Particularly, both UMAP and t-SNE produce more readable visualisations than MDS. Furthermore, both techniques

are able to capture and display data points with the same x,y coordinates while MDS does not do that (unless external jittering is used), which can lead to misleading interpretation of the test repository. Lastly, UMAP and t-SNE capture anomalies a bit differently. t-SNE creates spherical shapes, whereas UMAP creates more spaced clusters, where the points are seemingly randomly assigned around a fixed point.

One advantage of UMAPs over t-SNE is that the parameters are more comprehensible. The two parameters for UMAP are *number of neighbours* and *minimum distance*. A number of neighbours in this case simply means how many neighbours each point should consider as part of the "local neighbourhood" or in other words, cluster. Minimum distance controls the structure of the points such that higher value yield larger spacing between the points, while lower values preserve the local structure of the points and thus have less spacing. As UMAP is easier to use than t-SNE and also creates satisfactory visualisations, with nice compositions of the data points and clear clustering of the points, we recommend UMAP. Moreover, UMAP is also more modern and, according to L. McInnes et al. [22], performs faster and preserves local structure better than t-SNE.

5.3 Designing visualisation dashboards for testers

In our study, we performed an interview and asked practitioners and students with backgrounds in software engineering and testing about their preferences when it comes to visualising diversity. The answers were varying but there were points on which the majority of our participants agreed on. Based on these points we can confidently offer some guidance to practitioners that want to design dashboards to visualise test diversity.

Regarding filtering, sliders tend to be less useful and used in general. The slider can be useful when presenting the information to a less technically inclined person. Then the best way to present the slider is with the use of percentages since percentages are relatable to everyone. When it comes to using a slider to filter fault coverage (e.g., moving the slider shows test cases that cover a specific bug), most of our participants still preferred an interactive legend box. The general response was that it is nicer to see the whole picture of the graph first and then filter among one or more faults, instead of one by one.

The use of colour is important when creating a visualisation. The colour palette should consist of emotional colours such as a "good" colour indicating, for example, tests that find no faults and a "bad" colour indicating for example test finding faults. Accessibility is important, particularly since green and red are the default palette for test results, yet they are not colourblind friendly. We recommend designers include alternative palettes to cater for colourblind testers. When it comes to gradient versus discrete colours the participants of the interview agree that discrete colours are more comprehensible. The feedback that we have gotten indicates that colours also helps with decision making in some cases. In conclusion, when developing a dashboard that displays diversity, the developers should really have a bigger discus-

sion when it comes to the colours they use and definitely use discrete colours.

The most pleasing aspect of the visualisation was connecting the discrete colours with another visualisation technique. For instance, participants were positive in using the size of a data point to convey how many faults a test case has found. In addition, the fact that there were only two colours red and black (red indicating faults found and black indicating no faults found) made the visualisation clearer and more readable. This combination, by far, received the most positive feedback. The takeaway here is to have as few colours as possible and combine them with another visual technique to convey information.

Jittering and opacity of data points yielded more divided views. Some participants liked it, some thought that it made the graphs more confusing since there is so much data on the graphs already. In general, the answers were more skewed to having the jittering and opacity enabled because they helped reveal some hidden information, such as overlapping points. The opacity also made closely packed data points more visible. Here, we recommend the developers of potential dashboards for test diversity to leave the jittering and opacity options enabled by default but offering an option to disable them dynamically.

5.4 Threats to Validity

Here, we discuss the threats of validity that concerns our study. We will discuss internal validity, external validity and construct validity and conclusion validity in order.

5.4.1 Internal Validity

Our internal validity is threatened by the fact that, while collecting data, the repositories that we chose had many different versions with a varying number of mutants. In order to mitigate the issues in collecting data on faults/mutants, we chose specific versions of each project that had a good balance between a high number of mutants and tests to run our experiment.

Another threat to our internal validity is that we had very few participants for our interview. This is because finding participants with availability to participate in our study was challenging due to the pandemic. Ideally, we would perform the interviews with more testers with industry experience, however, research question 3 focuses on the usage of the dashboards, which is not necessarily related to industry experience. Moreover, the participants with academic background offered both similar and complementary views as those offered by practitioners working in the industry.

Furthermore, maturity was another threat to our internal validity during the interviews. Maturity, in short, means how time can affect the interview participants both negatively (bored, exhausted) and positively (learning) and, consequently, the

risk to skew the results [41]. We mitigate this threat by shuffling the order in which we present the visualisations per task so that each unique participant would see the sets of visualisations in different orders (e.g., one participant would first see the jittering enabled, whereas another would begin by seeing jittering disabled).

5.4.2 External Validity

A threat to external validity is that we cannot generalise our results due to the objects under test being outdated and generally small, particularly, when compared to the modern software systems. On the other hand, the focus on smaller systems allowed us to control the values of input and output and also avoid costly execution times. Additionally, the feasibility of running a larger system, multiple times on several artefacts is very low due to the computational power available (e.g., Levenshtein took 1.5 hours to run on the Grep project).

The risk of bias is introduced due to the system under tests selected from SIR, which affect our choice of metrics to use since we can only measure on what the repository has instrumented, i.e. APFD, coverage. Therefore, our experiments are less flexible in using different measures or we would introduce a risk of introducing faults in the SUTs by manipulating the source code. On the other hand, we can compare with already done research [19, 5] as a sanity check of our own results to some degree.

5.4.3 Construct Validity

Our construct validity is threatened by the fact that we have used two string-based and one compression distance function and haven't included any numerical type functions. This is because we wanted to be versatile for the different types of inputs in the projects under test, particularly for more complex inputs such as the XML files from NanoXML or the parsed outputs from Grep and Sed.

Another threat to our construct validity is the choice of dimensionality reduction techniques namely MDS, t-SNE and UMAP. We used these specific dimensionality reduction techniques because they were used in previous work related to testing and clustering [1, 21]. Although UMAP was not yet used in the software engineering field to our knowledge, it shows promise in other fields such as biotechnology [23].

In this thesis clustering was mainly used as a supporting tool for the dimensionality reduction. Having said that, there still exists some threats our construct validity by choosing to incorporate it to our study. The construct validity is threatened by only using one specific clustering technique namely DBSCAN. We used DBSCAN because, as stated in Chapter 2, it was used in many different studies and, in general, is quite popular with researchers across different academic disciplines. Another reason why we chose it was that, although dated, it has parameters that are easy to understand. The reason why we only used one technique is to focus our comparison on the dimensionality reduction techniques. Consequently, we avoid that risk that

using more clustering techniques increases the scope of our analysis to more levels, such that the comparison of the similarity maps becomes overwhelming.

5.4.4 Conclusion Validity

For RQ1, we mitigate the risk of statistical analysis by performing non-parametric tests due to our sample size being small. Since we could not reject our null hypothesis (Kruskals-Wallis statistical tests), there was no need to run a pair-wise post-hoc analysis (e.g., pairwise Wilcoxon signed-rank test with alpha corrections). In RQ1, we also use a visual analysis and to mitigate the risk of misinterpreting the data, we focus on the coarse-grained information provided by the different visualisation (heatmaps and histograms). This is done so that we specifically can examine the differences between each distance function, using the visualisations. Moreover, we align our visual findings with the correlation scores from the Mantel tests.

In RQ2, we also do a visual analysis and to mitigate similar risks, we constrain our scope to look at specifically the structural parts of the visualisations, i.e., where the clusters are, how the clusters look and the compositions of the clusters. By doing this, we can compare the three different techniques directly using the visualisations that were created.

Finally, in RQ3, we mitigate bias in our thematic analysis by triangulating our coding among us and our supervisor. Moreover, we limit our analysis to selective coding in order to delimit the number of themes to generate and, hence, focus on the benefits provided by the dashboards in connection to our proposed tasks.

5.5 Future Work

Our experiment was carried out using input and output artefacts. A promising extension to our work is to reproduce our analysis for code artefacts, in comparison to input and output artefacts. Code artefact would be the specific test script (code) that is used to execute the tests. This offers testers another option in their toolkit to apply test similarity, as the availability and challenges of extracting these different types of artefacts (input, output, test code) vary depending on the project.

Another suggestion is to expand our study to include modern software repositories, preferable in the industry. As stated in our threats to validity we used an older set of repositories of which the codebases were not representative of how programs look today. Particularly, contrasting those conclusions at different levels of tests (unit, integration or system) can offer insights into the versatility of diversity as a test measure.

Finally, as previously mentioned it would be nice to see an experiment like this carried out specifically for industry, in industry. This would be to further analyse the usage of diversity and how to use it as a tool with different visualisations. Since diversity is quite an intuitive technique, it would be nice to see how a manager would

use the visualisations versus how a developer or tester uses the visualisations, since both have different priorities or interests when it comes to testing decisions (e.g., test scope vs. modification coverage). Also to introduce industry into diversity as a whole, since it does not seem to be used in industry, even though it is quite an obvious aspect of programming.

6

Conclusion

In this thesis, we investigated different aspects regarding artefact diversity and software visualisation. Our investigation is based on different research methods such as an experimental study supported by visual and statistical analysis and a qualitative study.

In our experiment, we compare 3 different distance functions (Jaccard, Levenshtein and NCD) and investigated their trade-offs. We found that the two string distances Jaccard and Levenshtein captured more nuanced distinctions between string-based test artefacts than NCD, which is to be expected. However, NCD did perform similarly to Jaccard and Levenshtein when the data was bigger, such that no statistically significant difference was detected. Jaccard is faster than both NCD and Levenshtein. NCD and Levenshtein perform at a similar time for small strings, however, NCD is much faster than Levenshtein on bigger data. Our recommendation for textual artefacts and calculating diversity is to run Jaccard, followed by NCD. When it came to what type of test artefact to use, input vs output, we did not measure any significant difference between the two, but it was more important that the test data, in our case strings, were long enough to be analysed properly.

We then compared the visualisation of 3 different dimensionality reduction techniques (MDS, t-SNE and UMAP) and one clustering technique (DBSCAN). By comparing the comprehensibility of the different graphs produced, we argue that t-SNE and UMAP both outperform MDS when it comes to producing visualisations. During our visual analysis, we concluded that the information conveyed by those two techniques is much clearer especially when it comes to the data that is more spread out. We also found that UMAP has hyperparameters that are easier to understand. This indicates that UMAP is a good choice for its users since the learning curve would not be that big and, additionally, UMAP outperforms t-SNE in terms of run-time complexity.

Finally, now that we have our visualisations that are readable to the human eye, we performed a qualitative analysis in which we conducted a couple of interviews, to see how software engineering practitioners work with our visualisation. From the resulting interviews, we concluded that the colour of the data has a big effect on decision making, even better if the colours are discrete and are coupled with a visual technique (such as making a test case's data point bigger if it found more faults). Furthermore, we found that other interactive visual techniques such as sliders are not used as much when performing tasks that are related to test selection but was

suggested by interviewees to be a nice tool that sparks discussions with less technically inclined people in the company.

We summarize our findings from above and conclude these points as improvements to a tester's toolkit by firstly, offering a trade-off in different test selection techniques supported by diversity. Secondly, bringing awareness to the characteristics of comprehensible visualisation of multidimensional data and, lastly, defining a baseline of properties to encourage the development and adoption of test analytics. One risk of the growing popularity of automated testing approaches is to forget the inclusion of an expert's opinion of viewpoint, whereas the combination of software and visualisation used in our thesis can bring the human back into the testing loop, hence, strengthening the synergies between automated and manual approaches.

Bibliography

[1] F. G. De Oliveira Neto, R. Feldt, L. Erlenhov, and J. B. D. S. Nunes, "Visualizing Test Diversity to Support Test Optimisation," in *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, pp. 149–158, 2018.

[2] F. Dobslaw, F. G. de Oliveira Neto, and R. Feldt, "Boundary Value Exploration for Software Analysis," *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, oct 2020.

[3] 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*, pp. 178–186, 2008.

[4] F. G. d. O. Neto, F. Dobslaw, and R. Feldt, "Using mutation testing to measure behavioural test diversity," in *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 254–263, 2020.

[5] R. Feldt, S. Poulding, D. Clark, and S. Yoo, "Test Set Diameter: Quantifying the Diversity of Sets of Test Cases," in *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pp. 223–233, 2016.

[6] E. G. Cartaxo, P. D. Machado, and F. G. O. Neto, "On the use of a similarity function for test case selection in the context of model-based testing," *Software Testing, Verification and Reliability*, vol. 21, no. 2, pp. 75–100, 2011.

[7] P. Jaccard, "Etude de la distribution florale dans une portion des alpes et du jura," *Bulletin de la Societe Vaudoise des Sciences Naturelles*, vol. 37, pp. 547–579, 01 1901.

[8] F. G. de Oliveira Neto, A. Ahmad, O. Leifler, K. Sandahl, and E. Enoiu, "Improving continuous integration with similarity-based test case selection," in *Proceedings of the 13th International Workshop on Automation of Software Test*, pp. 39–45, 2018.

[9] F. G. de Oliveira Neto, R. Torkar, and P. D. Machado, "Full modification coverage through automatic similarity-based test case selection," *Information and Software Technology*, vol. 80, pp. 124–137, 2016.

[10] A. E. V. B. Coutinho, E. G. Cartaxo, and P. D. de Lima Machado, "Analysis of distance functions for similarity-based test suite reduction in the context of model-based testing," *Software Quality Journal*, vol. 24, no. 2, pp. 407–445, 2016.

[11] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, vol. 10, pp. 707–710, Soviet Union, 1966.

[12] J. B. Kruskal, "An overview of sequence comparison: Time warps, string edits, and macromolecules," *SIAM review*, vol. 25, no. 2, pp. 201–237, 1983.

[13] B. Kessler, "Computational dialectology in irish gaelic," *arXiv preprint cmp-lg/9503002*, 1995.

[14] C. Landing, S. Tahvili, H. Haggren, M. Langkvis, A. Muhammad, and A. Loufi, "Cluster-based parallel testing using semantic analysis," in *2020 IEEE International Conference On Artificial Intelligence Testing (AITest)*, pp. 99–106, IEEE, 2020.

[15] M. Li, X. Chen, X. Li, B. Ma, and P. M. Vitányi, "The similarity metric," *IEEE transactions on Information Theory*, vol. 50, no. 12, pp. 3250–3264, 2004.

[16] C. H. Bennett, P. Gács, M. Li, P. M. Vitányi, and W. H. Zurek, "Information distance," *IEEE Transactions on information theory*, vol. 44, no. 4, pp. 1407–1423, 1998.

[17] M. Li, P. Vitányi, *et al.*, *An introduction to Kolmogorov complexity and its applications*, vol. 3. Springer, 2008.

[18] N. Tran, "The normalized compression distance and image distinguishability," in *Human Vision and Electronic Imaging XII*, vol. 6492, p. 64921D, International Society for Optics and Photonics, 2007.

[19] C. Henard, M. Papadakis, M. Harman, Y. Jia, and Y. Le Traon, "Comparing white-box and black-box test prioritization," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pp. 523–534, 2016.

[20] V. Sumithra and S. Surendran, "A review of various linear and non linear dimensionality reduction techniques," *Int. J. Comput. Sci. Inf. Technol*, vol. 6, pp. 2354–2360, 2015.

[21] S. Tahvili, L. Hatvani, M. Felderer, W. Afzal, and M. Bohlin, "Automated Functional Dependency Detection Between Test Cases Using Doc2Vec and Clustering," in *2019 IEEE International Conference On Artificial Intelligence Testing (AITest)*, pp. 19–26, IEEE, 2019.

[22] L. McInnes, J. Healy, and J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction," 2020.

[23] E. Becht, L. McInnes, J. Healy, C.-A. Dutertre, I. W. Kwok, L. G. Ng, F. Ginhoux, and E. W. Newell, "Dimensionality reduction for visualizing single-cell data using umap," *Nature biotechnology*, vol. 37, no. 1, pp. 38–44, 2019.

[24] W. S. Torgerson, "Multidimensional scaling: I. theory and method," *Psychometrika*, vol. 17, no. 4, pp. 401–419, 1952.

[25] M. A. Cox and T. F. Cox, "Multidimensional scaling," in *Handbook of data visualization*, pp. 315–347, Springer, 2008.

[26] M. C. Hout, M. H. Papesh, and S. D. Goldinger, "Multidimensional scaling," *Wiley Interdisciplinary Reviews: Cognitive Science*, vol. 4, no. 1, pp. 93–103, 2013.

[27] L. Van Der Maaten, E. Postma, and J. Van den Herik, "Dimensionality reduction: a comparative," *J Mach Learn Res*, vol. 10, no. 66-71, p. 13, 2009.

[28] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne.," *Journal of machine learning research*, vol. 9, no. 11, 2008.

[29] G. Hinton and S. T. Roweis, "Stochastic neighbor embedding," in *NIPS*, vol. 15, pp. 833–840, Citeseer, 2002.

[30] M. Wattenberg, F. Viégas, and I. Johnson, "How to use t-sne effectively," *Distill*, 2016.

[31] K. Fukunaga and D. R. Olsen, "An algorithm for finding intrinsic dimensionality of data," *IEEE Transactions on Computers*, vol. 100, no. 2, pp. 176–183, 1971.

[32] J. C. Bezdek, R. Ehrlich, and W. Full, "Fcm: The fuzzy c-means clustering algorithm," *Computers & geosciences*, vol. 10, no. 2-3, pp. 191–203, 1984.

[33] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise.," in *Kdd*, vol. 96, pp. 226–231, 1996.

[34] M. Hahsler, M. Piekenbrock, and D. Doran, "dbscan: Fast density-based clustering with r," *Journal of Statistical Software*, vol. 91, no. 1, pp. 1–30, 2019.

[35] L. Bedu, O. Tinh, and F. Petrillo, "A tertiary systematic literature review on software visualization," in *2019 Working Conference on Software Visualization (VISSOFT)*, pp. 33–44, IEEE, 2019.

[36] F. G. de Oliveira Neto, M. Jones, and R. d. S. Martins, "Visualisation to Support Fault Localisation in Distributed Embedded Systems within the Automotive Industry," in *2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pp. 112–117, 2018.

[37] J. Slater, C. Anslow, J. Dietrich, and L. Merino, "Corpusvis–visualizing software metrics at scale," in *2019 Working Conference on Software Visualization (VISSOFT)*, pp. 99–109, IEEE, 2019.

[38] S. P. Reiss, "The paradox of software visualization," in *3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis*, pp. 1–5, IEEE, 2005.

[39] A. Dix and G. Ellis, "Starting simple: adding value to static visualisation through simple interaction," in *Proceedings of the working conference on Advanced visual interfaces*, pp. 124–134, 1998.

[40] R. Feldt, M. Staron, E. Hult, and T. Liljegren, "Supporting Software Decision Meetings: Heatmaps for Visualising Test and Code Measurements," in *2013 39th Euromicro Conference on Software Engineering and Advanced Applications*, pp. 62–69, 2013.

[41] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[42] H. Do, S. G. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact.," *Empirical Software Engineering: An International Journal*, vol. 10, no. 4, pp. 405–435, 2005.

[43] B. Miranda, E. Cruciani, R. Verdecchia, and A. Bertolino, "Fast approaches to scalable similarity-based test case prioritization," in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pp. 222–232, IEEE, 2018.

[44] K.-J. Stol, P. Ralph, and B. Fitzgerald, "Grounded theory in software engineering research: a critical review and guidelines," in *Proceedings of the 38th International Conference on Software Engineering*, pp. 120–131, 2016.

A

Appendix 1

A.1 Grep dimensionality reduction visualisation

A.1.1 Grep - MDS

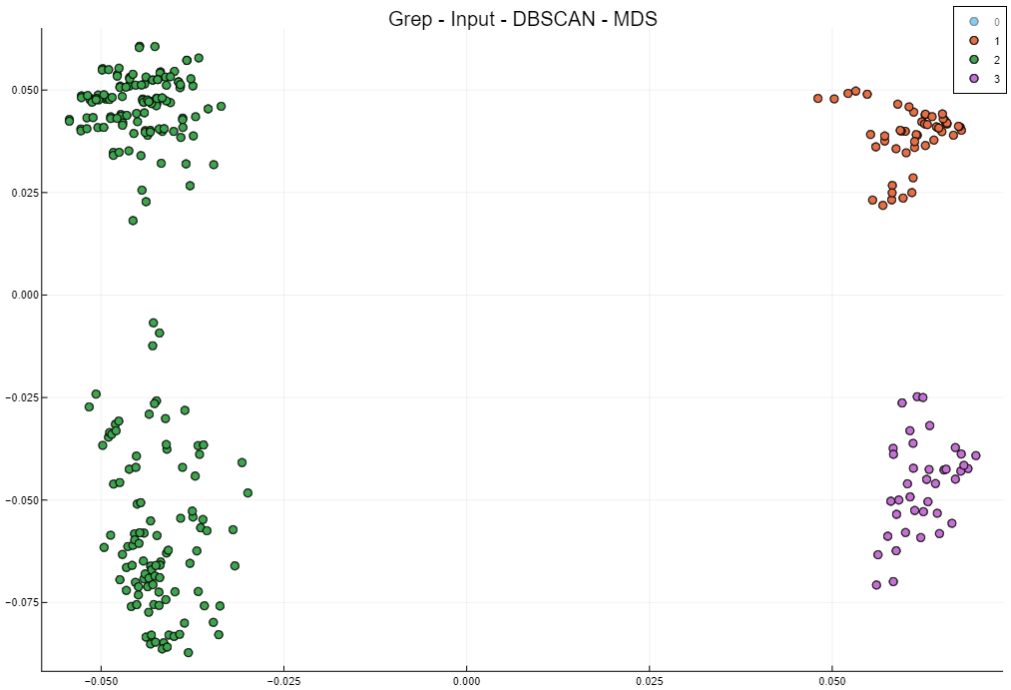

Figure A.1: Dimensionality reduction MDS on the input data of Grep repository using DBSCAN clustering

A.1.2 Grep - TSNE

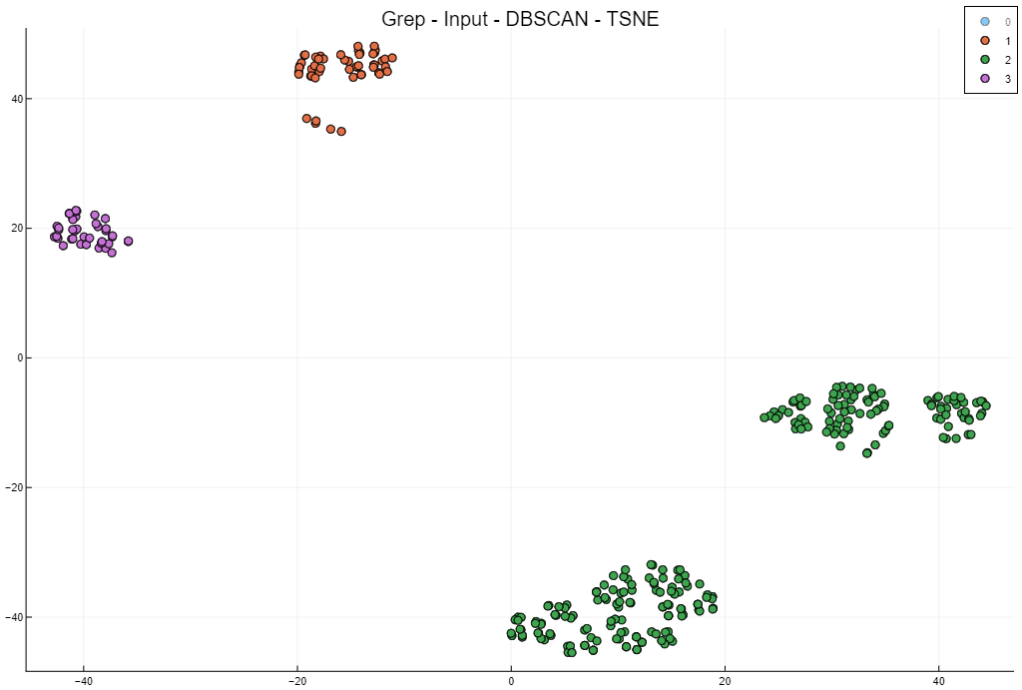

Figure A.2: Dimensionality reduction TSNE on the input data of Grep repository using DBSCAN clustering

A.1.3 Grep - UMAP

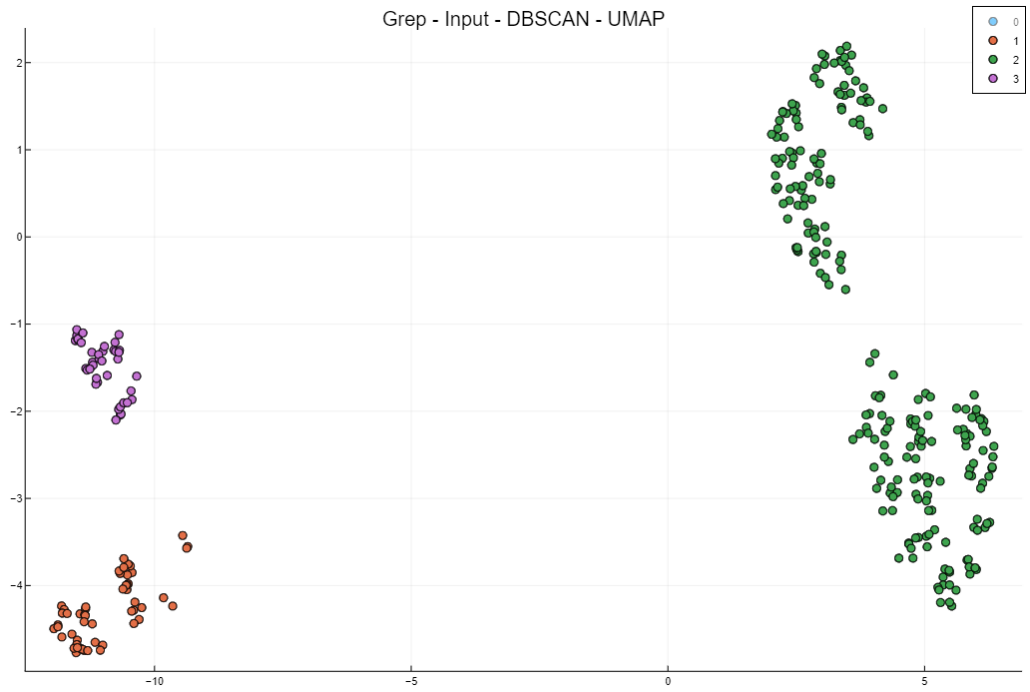

Figure A.3: Dimensionality reduction UMAP on the input data of Grep repository using DBSCAN clustering

A.2 Research Question 3 - Similarity Maps

A.2.1 Set One

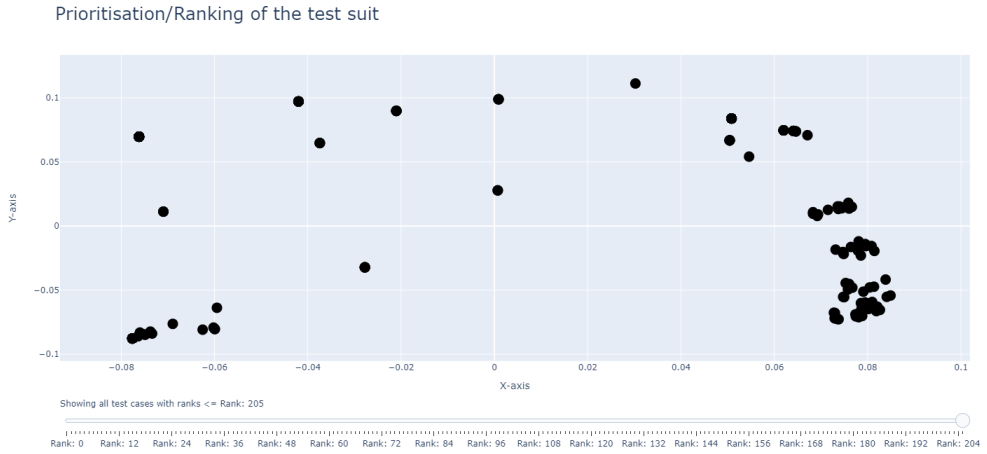

Figure A.4: Visualisation with jittering and opacity off

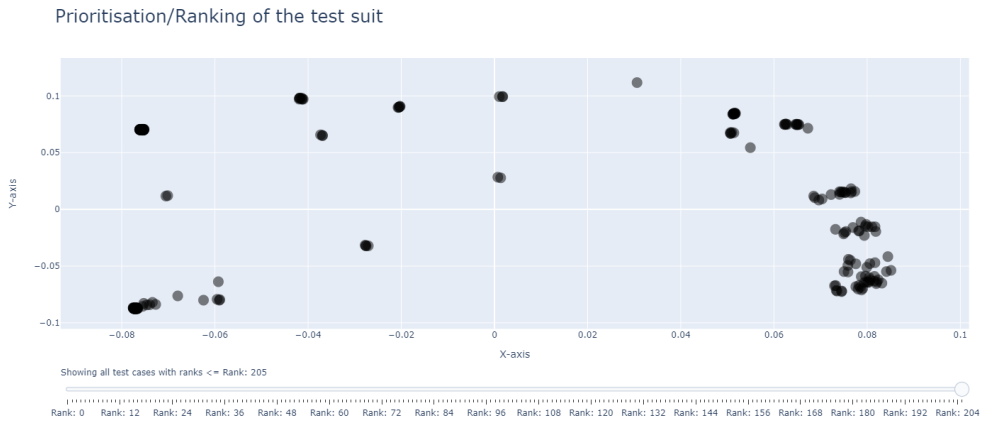

Figure A.5: Visualisation with jittering and opacity on

A.2.2 Set Two

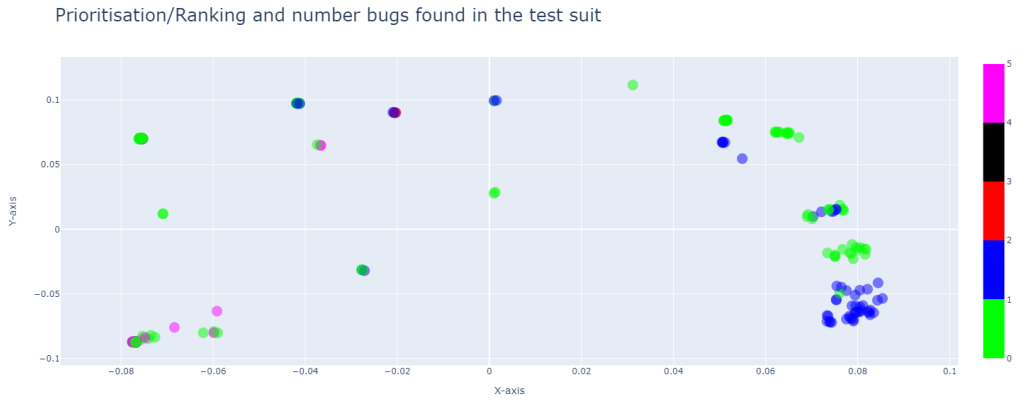

Figure A.6: Visualisation with discrete colours

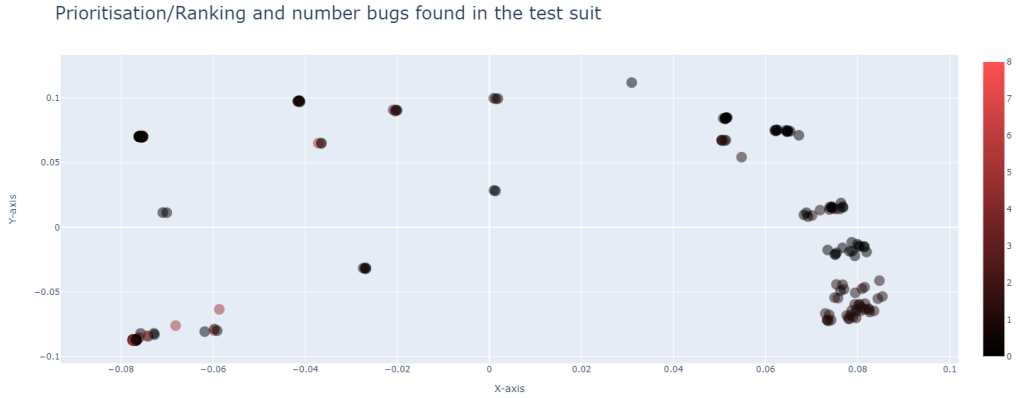

Figure A.7: Visualisation with gradient colours

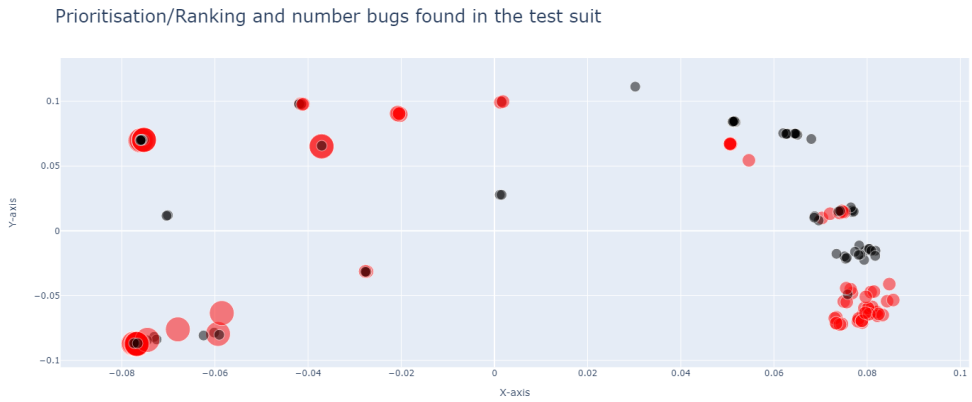

Figure A.8: Visualisation with discrete colours and differing sizes of data points

A.2.3 Set Three

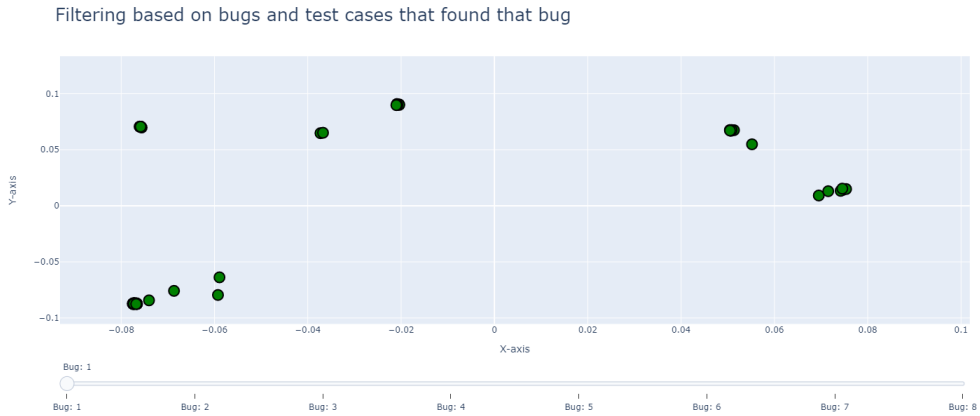

Figure A.9: Visualisation with slider as a method of filtering test cases per bug

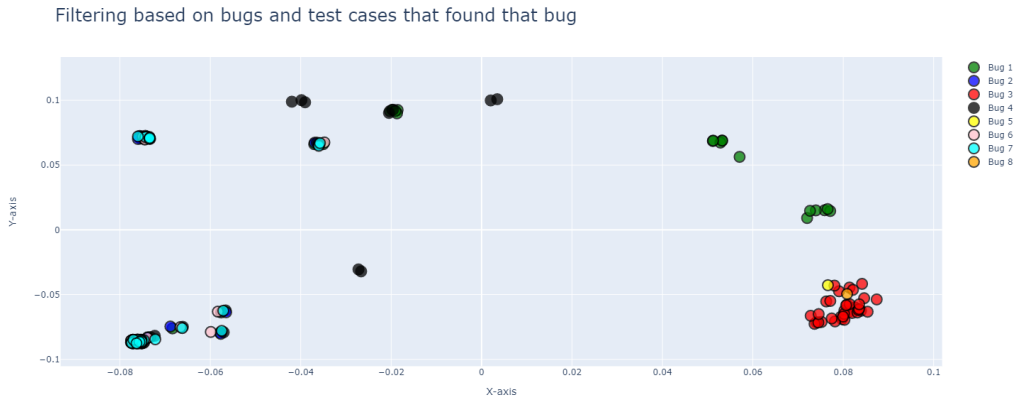

Figure A.10: Visualisation with a clickable legend as a method of filtering test cases per bug