Application of Topic Models for Test Case Selection
- A comparison of similarity-based selection techniques

Tillämpning av ämnesmodeller för testfallsselektion

Kim Askling

Supervisor : Azeem Ahmad
Examiner : Ola Leffler

External supervisor : Magnus Söderlind
Upphovsrätt

Detta dokument hålls tillgängligt på Internet - eller dess framtida ersättare - under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/.

Copyright

The publishers will keep this document online on the Internet - or its possible replacement - for a period of 25 years starting from the date of publication barring exceptional circumstances.

The online availability of the document implies permanent permission for anyone to read, to download, or to print out single copies for his/hers own use and to use it unchanged for non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional upon the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: http://www.ep.liu.se/.

© Kim Askling
Abstract

Regression testing is just as important for the quality assurance of a system, as it is time consuming. Several techniques exist with the purpose of lowering the execution times of test suites and provide faster feedback to the developers, examples are ones based on transition-models or string-distances. These techniques are called test case selection (TCS) techniques, and focuses on selecting subsets of the test suite deemed relevant for the modifications made to the system under test.

This thesis project focused on evaluating the use of a topic model, latent dirichlet allocation, as a means to create a diverse selection of test cases for coverage of certain test characteristics. The model was tested on authentic data sets from two different companies, where the results were compared against prior work where TCS was performed using similarity-based techniques. Also, the model was tuned and evaluated, using an algorithm based on differential evolution, to increase the model’s stability in terms of inferred topics and topic diversity.

The results indicate that the use of the model for test case selection purposes was not as efficient as the other similarity-based selection techniques studied in work prior to this thesis. In fact, the results show that the selection generated using the model performs similar, in terms of coverage, to a randomly selected subset of the test suite. Tuning of the model does not improve these results, in fact the tuned model performs worse than the other methods in most cases. However, the tuning process results in the model being more stable in terms of inferred latent topics and topic diversity. The performance of the model is believed to be strongly dependent on the characteristics of the underlying data used to train the model, putting emphasis on word frequencies and the overall sizes of the training documents, and implying that this would affect the words’ relevance scoring to the better.
Acknowledgments

I would like to thank Altran Karlstad for giving me the opportunity to conduct my work at their organization. Thanks to everyone, who welcomed and made me feel as a part of the team. A special thanks to my supervisor at the company, Magnus Söderlind, for always being there when I needed to air some ideas and for always reminding me to "keep it realistic".

I would also like to thank Azeem Ahmad and Ola Leifler, my supervisor and examiner at Linköping University for helping me with the definition of the thesis scope, as well as good discussions regarding the technical approaches.

Finally, I would like to thank my opponent Martin Lundberg for providing good and thorough feedback on both this report and the presentation, and thus helping me improve their quality and understandability.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>iii</td>
</tr>
<tr>
<td>Acknowledgments</td>
<td>iv</td>
</tr>
<tr>
<td>Contents</td>
<td>v</td>
</tr>
<tr>
<td>List of Figures</td>
<td>vii</td>
</tr>
<tr>
<td>List of Tables</td>
<td>viii</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Motivation</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Aim</td>
<td>2</td>
</tr>
<tr>
<td>1.3 Research questions</td>
<td>2</td>
</tr>
<tr>
<td>1.4 Delimitations</td>
<td>3</td>
</tr>
<tr>
<td>2 Theory</td>
<td>4</td>
</tr>
<tr>
<td>2.1 Related work</td>
<td>4</td>
</tr>
<tr>
<td>2.2 Software testing</td>
<td>7</td>
</tr>
<tr>
<td>2.3 Topic modelling</td>
<td>9</td>
</tr>
<tr>
<td>2.4 Dirichlet distribution</td>
<td>10</td>
</tr>
<tr>
<td>2.5 Latent dirichlet allocation</td>
<td>13</td>
</tr>
<tr>
<td>3 Method</td>
<td>16</td>
</tr>
<tr>
<td>3.1 Data sets</td>
<td>16</td>
</tr>
<tr>
<td>3.2 Preprocessing</td>
<td>17</td>
</tr>
<tr>
<td>3.3 Topic modelling</td>
<td>18</td>
</tr>
<tr>
<td>3.4 Test case selection</td>
<td>19</td>
</tr>
<tr>
<td>3.5 Parameter tuning</td>
<td>19</td>
</tr>
<tr>
<td>3.6 Topic model evaluation</td>
<td>22</td>
</tr>
<tr>
<td>3.7 Visualization</td>
<td>23</td>
</tr>
<tr>
<td>4 Results</td>
<td>25</td>
</tr>
<tr>
<td>4.1 Preprocessing</td>
<td>25</td>
</tr>
<tr>
<td>4.2 Model stability</td>
<td>25</td>
</tr>
<tr>
<td>4.3 Test case selection</td>
<td>31</td>
</tr>
<tr>
<td>5 Discussion</td>
<td>35</td>
</tr>
<tr>
<td>5.1 Results</td>
<td>35</td>
</tr>
<tr>
<td>5.2 Method</td>
<td>37</td>
</tr>
<tr>
<td>5.3 The work in a wider context</td>
<td>40</td>
</tr>
<tr>
<td>6 Conclusion</td>
<td>42</td>
</tr>
</tbody>
</table>
# List of Figures

2.1 The V-model visualizing the software development levels .................................. 8
2.2 Topic model factorization of a corpus $C$ where $\Theta$ is the distribution of documents over topics and $\phi$ is the distribution of topics over words .......................................................... 10
2.3 The plane that spans the values of the 2-simplex .................................................. 11
2.4 Dirichlet probability distributions, $\theta$, being a 2-simplex, with different values on $\alpha$ .................................................................................................................. 12
2.5 Graphical model representation of LDA using plate notation. Plate $M$ denotes repetitions for each document in the corpus and plate $N$ denote repetitions for the distribution of topics and words within each document .............................................. 13
3.1 The work flow for creating dictionary and BoW representation .................................. 18
3.2 Graphical model visualizing the calculation of the raw score ..................................... 23
4.1 The raw scores for the default and tuned surveillance company models ....................... 26
4.2 The delta score between the surveillance company models’ raw scores ....................... 26
4.3 The raw scores for the default and tuned automotive company models ....................... 27
4.4 The delta score between the automotive company models’ raw scores ....................... 27
4.5 Topic diversity between two surveillance company models with different corpus order, using default configuration ................................................................. 28
4.6 Topic diversity between two surveillance company models with different corpus order, using tuned configuration with $k = 223$ .......................................................... 29
4.7 Topic diversity between two surveillance company models with different corpus order, using tuned configuration with $k = 109$ .......................................................... 29
4.8 Topic diversity between two automotive company models with different corpus order, using default configuration ................................................................. 30
4.9 Topic diversity between two automotive company models with different corpus order, using tuned configuration with $k = 237$ .......................................................... 30
4.10 Coverage of features for selections of surveillance company test suite .................. 31
4.11 Coverage of dependencies for selections of surveillance company test suite ............ 32
4.12 Total execution time of surveillance company selection .......................................... 32
4.13 Coverage of test steps of automotive company selection ........................................ 33
4.14 Coverage of test steps of automotive company selection, when only selecting test cases containing test steps ................................................................. 33
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Summary of data sets from each company</td>
<td>16</td>
</tr>
<tr>
<td>3.2</td>
<td>Parameters tuned in this thesis project</td>
<td>21</td>
</tr>
<tr>
<td>4.1</td>
<td>Results of corpus sizes before and after filtering is applied</td>
<td>25</td>
</tr>
<tr>
<td>4.2</td>
<td>Performance of techniques compared to respective random selection for the</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>surveillance company’s test suite, measured in %. Grouped in two for clarifi-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>cation of different random selections</td>
<td></td>
</tr>
<tr>
<td>A.1</td>
<td>Performance of the LDA model, with default and tuned configurations, compared</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>to the random selection used in the thesis</td>
<td></td>
</tr>
<tr>
<td>A.2</td>
<td>Performance of the three similarity-based techniques compared to the random</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>selection used by de Oliveira Neto et al.</td>
<td></td>
</tr>
</tbody>
</table>
Listings

3.1 Pseudo code for LDADE algorithm ................................................................. 20
3.2 Pseudo code for extrapolating algorithm ....................................................... 20
3.3 Pseudo code for calculating similarity score between models (raw score) ...... 21
1 Introduction

This chapter presents the motivation behind the thesis and formulates the research questions and delimitations.

1.1 Motivation

One of the major costs during the development phase of a product is the testing activities, up to 50% of software development costs may be needed for the software testing [1]. This activity may include the so-called regression testing of the system under test (SUT), which is the re-testing of software in order to make sure that previous functioning parts of the system is still functioning after modification being made to the software.

Due to the highly repetitive and time consuming process that is regression testing, the area has been a popular subject for both optimization and automation research as in [2] and [3]. One particular technique used with regression testing is test case selection (TCS), where a subset of the test suite is selected in order to strive for certain criteria. These criteria can be based on the lowering of test suites execution times, the criticality of test cases or other perceived effectiveness. One example would be to only use 70% of the test suite with the objective to minimize the run time of the test session, and with the desire to not compromise the portion of the SUT being tested. Researchers have explored TCS using several methods to perform the selection of the subset of test cases. These methods may be based on so called dynamic techniques, where the execution information of test cases is analyzed such as number of faults detected or the portion of the SUT that is tested. One example of a dynamic technique is adaptive random testing where test cases are randomly selected from a set of candidates, which is incrementally updated with each executed test case, in order to provide an even spread of executed test cases [3]. Another dynamic technique, based on a greedy heuristic, is to select the test cases that cover as large portion of the SUT that has not yet been covered by previous test cases, thus implementing the “next best” philosophy as in [5] and [6].

However, to address the situations where the execution information is not available and only specification models of the test cases exist, the TCS may be based on static techniques. A static technique implies that the test cases are not executed but instead analyzed by their descriptions and definitions, such as source code. One example of such static techniques are the ones based on similarities of the test cases. It could mean that the test cases are selected
1.2. Aim

Based on their similarities in coverage of execution-paths within the SUT, as in [7] and [8], or based on the textual similarities of their descriptions (source code) as in [9].

Other static techniques that have been exercised thoroughly in TCS-research are based on modelling software with the help of underlying "topics" hidden in the descriptions, a process called topic modelling. One popular topic model technique is Latent Dirichlet Allocation (LDA) which is a probabilistic model that makes use of the linguistic data within a set of documents and infers the underlying latent (hidden) topic distributions [10]. Earlier work with topic models includes tasks such as identifying people's roles in a company based on topics within emails along with their sender-receiver information [11], and semantic clustering of source code using identifier names, comments and string literals in order to improve software comprehension [12]. Topic models have also been used for test case selection [13] and for classification of test cases without selections [14], implying that the hidden topics in the source code may be used for identifying similarities in test cases.

It is interesting to evaluate different TCS techniques, using the same data and compare the results for each technique against the other, to find the optimal technique for that specific data set [15]. Even more interesting if authentic data is used, meaning that the data consist of artifacts used by a company instead of mock-up data with artificial bugs inserted for learning purposes. This is why the data used by de Oliveira Neto et al. in [9] has been made available for this thesis project, with the intent of having the results compared and to find out if the use of topic models proves better than the other techniques. De Oliveira Neto et al. investigated several similarity-based techniques for TCS, with the intent of reducing feedback times on integration test activities in a continuous integration environment. The comparison is based on coverage of certain characteristics of the data sets, described further in section 3.1.

Along with this comparison, it is also interesting to see how the tuning of LDA-configurations may cause improved stability for the model and how the inference of topic distributions is affected. Agrawal et al. emphasize the need for tuning LDA-parameters due to its non-deterministic stochastic sampling process, and the lack of considerations for tuning in previous work [16]. They present their results after utilizing a tuning algorithm, LDA Differential Evolution (LDADE), to tune the model’s parameters and conclude that the stability of the model improves dramatically when tuned, compared to using the default parameters of the model.

1.3 Research questions

These research questions have taken form to guide the project according to the aim:

RQ1: What is the effect of using the LDA model for TCS, as compared to the similarity-based techniques used in previous work by de Oliveira Neto et al. in [9]?
RQ2: What is the effects of parameter tuning using LDADEN on the topic model in order to increase topic stability and topic diversity?

Topic stability refers to the concept of the model being able to deduce the same, or very similar, topics independent of the input order of the documents (test cases). Topic diversity refers to the observed differences between the deduced topics, i.e. the differences in terms associated with respective topic before and after tuning.

1.4 Delimitations

This thesis project is conducted on two data sets from two companies associated with Software Center. The same data sets were used in previous work by Oliveira Neto et al., with the sole purpose of comparing the results from this thesis project with the results from their work. The units of analysis are thus the same as for de Oliveira Neto et al. and are described further in 3.1.

Finally, due to limitations in time, tuning of the model is only done with the aim to improve stability of the LDA model. No tuning is done with respect to the results of the selection of test cases.

1https://www.software-center.se
This chapter presents related work as well as the theory needed to answer the research questions with good support and validity.

2.1 Related work

This section presents the related work used in this thesis project as background as well as a source for the covered theory.

Test case selection

Test case selection has been an area of interest for some time now, with researchers investigating techniques to minimize execution times of regression tests [17]. One common technique, nowadays mostly used as a reference technique, is to select a subset by randomly selecting test cases from a test suite until the desired size is reached. This is a simple, yet effective, approach to implement when wanting to select a diverse subset [18]. An extension to this is adaptive random testing that seeks to create a more evenly spread selection based on the input space [4]. The creators of the technique, Chen et al., describe the technique as a random test selection technique that takes the patterns of failure-causing inputs into consideration. It tries to create an even spread of inputs to the system, where the next test cases are selected at random from a candidate-list containing test cases that has a large distance to the input set already tested. This technique proved useful in order to create a test selection based on diversity. Common to these techniques is that they, in some sense, use similarities between test cases as a foundation for the ordering.

De Oliveira Neto et al. use coverage of test requirements, test dependencies and test steps with a similarity-based approach for their selection of test cases [9]. They use a static black box technique where the only data available is the test suite of a system along with the individual test case coverage. The black box technique implies that there is no information regarding the actual SUT, only of the test cases. Their approach aims to select a subset of the test suite that is as diverse as possible in terms of coverage, providing developers with quicker feedback. The authors take a stand for desiring a diverse test case selection that is able to exercise multiple yet distinct parts of the SUT, thus allowing for removal of redundant test cases. For the comparison, the authors use four selection techniques where the first
is a random selection that acts as a reference, the remaining three are based on similarity functions: Normalized Levenshtein (NL), Jaccard Index (JI) and Normalized Compression Distance (NCD). NL focuses on the edit distance between two strings, meaning the distance in number of edit operations needed for similarity [19]. JI measures the similarity of two sample sets and is simply explained as the intersection over union of two samples [19]. NCD is a more general distance metric that measures the edit distance between two objects transformed to strings of 0s and 1s, meaning that it can even measure the distance of two objects of different types (such as a program and a picture) [20]. The results of the case study show that a reduction in time of up to 92% is achievable when using the similarity based approach. They also show that they are able to provide full coverage of test requirements and dependencies after only selecting 15% and 35% of the original test suite, respectively. Whereas full coverage of test steps is achieved after reducing the test suite with up to 20%, followed by a coverage of 99.4% up to a reduction of 70%.

**Topic models in software**

Topic models have become a common technique to use on software in order to improve the comprehension of the system, meaning the system’s ability to be understood and interpreted correctly by developers. The linguistic data in the form of comments, identifier names, and string literals within the source code is used as the input corpus and a set of topics are inferred. These topics are then meant to represent different parts of the system, such as “audio controls” or “image rendering”.

Maskeri et al. used an approach based on the LDA model, together with some human assistance to identify domain topics from source code that are comprehensible to humans [21]. The authors implement a tool for extracting and labeling of domain topics based on the names of function elements, such as names of data structures and files, and comments in the code. Their implementation is able to extract a set of topics. However, to the authors disappointment, the tool was not able to automatically derive human interpretable labels of the topics that were satisfactorily enough. They conclude that their LDA-based model is able to identify some of the domain topics, but not all, and argue for the need for a domain expert that has the knowledge to manually analyze the resulting clusters of terms to see if they are good representations of the domain topics. In terms of parameter tuning, the authors identify the number of topics to be a major factor to the resulting topics. They also conduct an experiment to conclude the optimal number of topics for their specific data set. The parameters $\alpha$ and $\beta$, which represent the prior knowledge of the topic-document distribution and word-topic distribution respectively, are also identified as contributors to the result. However, these parameters are only explained to be varied to values that seem to improve the topic inference.

Prior to the work of Maskeri et al., another paper is written by Kuhn, Ducasse and Gîrba where Latent Semantic Indexing (LSI) is used for the same purpose of extracting domain topics from source code [12]. Apart from the difference in technique, the two papers uses two different definitions of the concept “topic”. Kuhn et al. consider topics, or linguistic topics as they call them, to be the groups (clusters) of semantically related source artifacts, such as names of packages and methods. Maskeri et al., on the other hand, consider topics to be the linguistic terms that they extract from the the identifier names and comments, not the names themselves. Kuhn et al. describe their aim to be to help developers get a better insight in new software by providing a better first impression of the system, revealing prior developers’ knowledge hidden in identifier names and comments and to enrich analysis of the software with this “informal” information. As a result they successfully manage to extract and cluster the linguistic data and use colorful distribution maps to visualize the semantic relationships between the software artifacts. They conclude that even though they were not able to successfully extract and label the topics every time, they are able to improve the first contact between a developer and a new system to the better. They also conclude that a contributing
factor to the results is the quality of the software, meaning that code with trivial identifier naming such as "method_1" or "base_class" is harder to analyze for semantic relationships.

There have also been specific tools developed that use topic models for the sole purpose of classifying the sub-components of a system. Such a tool is the Topic\textsubscript{XP} tool developed by Savage et al. and described in [22]. This tool is a plug-in for the Eclipse IDE and uses LDA for linguistic extraction with the purpose of visualizing concepts or features implemented in the classes of software. In general, the tool uses two views to aid the developer to easier grasp the underlying features of the software. The first view displays an overview of the topics with their most relevant terms, associated documents and dependencies between topics based on dependency-graphs of the software’s class level. The second view gives a more detailed insight of a selected topic and visualizes the most important documents for that topic in a tree-map\textsuperscript{1}. The paper also describes the process of having the tool tested by four participants tasked to perform a concept localization for four maintenance tasks with two different systems. The participants compared the usage of Topic\textsubscript{XP} against "regular" Eclipse IDE where they only used manual methods such as browsing files and following static dependencies. The authors conclude that Topic\textsubscript{XP}, and topic models in general, may indeed aid developers with visualizing the underlying concepts and features of a system.

Thomas et al. present, in their paper [13], a static black box test case prioritization (TCP) technique based on topic models of test cases’ linguistic data in the source code. This technique is similar to the one used in this thesis project, although the major difference is that they use their technique for TCP, and this thesis project focuses on TCS followed by a tuning process of the LDA-model’s parameters. For the model, Thomas et al. use the default parameters for $\alpha$ and $\beta$, being 0.01 in both cases, while using the topic number $k = N/2.5$, where N equals the total number of documents in the SUT. The authors use a technique to mitigate the randomness of the model by running the model-creation process multiple times with different random seeds for the input documents. They also perform a small sensitivity test to see how variations of the parameters changes the results of the prioritized test cases, measured as average percentage of faults detected (APFD) which is a metric for a test set’s ability to find faults quickly. For the parameters $< k,\alpha,\beta >$, they alter one parameter at a time, double and half it’s default value, keeping the rest of the parameters the same. They conclude that even though some minor deviations are noticed, they are not large enough to be considered as definitive for the results. The authors also present a comparison of their model against other static black box techniques in regards to their APFD results. Their comparison shows that their topic model technique is always at least as effective in finding faults quickly as the compared techniques, string-based prioritization, random prioritization and a black box version of a call-graph based prioritization. In the best case, their topic model technique outperforms the other techniques with 31%.

Topic model techniques have been subject of other comparison studies as well. One of these studies is made by Luo et al. in 2018 [15], which build on their earlier work from 2016 [23]. This study is, as the title of the article reveals, an extensive one where 58 different projects found on GitHub\textsuperscript{2} are used to evaluate a number of static and dynamic TCP techniques, whereas two of them are techniques based on topic modelling. The techniques are utilized both on test-class level and on test-method level, thus providing results for different granularity of the TCP. The techniques are evaluated using the APFD metric as well as a cost cognizant version, APFD\textsubscript{c}. During the study, the authors also inject mutations into the SUT using the PIT mutation tool\textsuperscript{3} and evaluate their effects on the final results. The study show that in general the static techniques outperform the dynamic techniques in the case of test-class granularity, which includes the ones based on topic models. On the other hand, for the case of test-method granularity, the dynamic techniques outperform the static techniques in terms of APFD. Out of the five static techniques tested, in the case of test-class granularity,
the results show that the two techniques based on topic models perform worse than the other static techniques. For the test-method case, however, the two topic model techniques perform better whereas the best of them comes second in ranking among the static techniques. The authors conclude that the topic model techniques differ in performance for different subject programs and especially in the case of test-class granularity. They also differ a lot between each other, implying that the implementation of the topic models matters. The authors also conclude that researchers should consider both the granularity level of the TCP as well as the characteristics of both the subject systems and the tools used. Furthermore, they conclude that subject size, evolution of software and quantity of injected mutants and their type makes no significant effect on their measures of the TCP effectiveness.

2.2 Software testing

Software testing is the concept of analyzing a test item, such as a piece of code or some other generated software artifact, to ensure its functionality and quality according to given requirements. There are several formal definitions of the testing concept. The ISO/IEC/IEEE International Standard, 29119, define testing as "a set of activities conducted to facilitate discovery and/or evaluation of properties of one or more test items" [24]. Another definition, stated by Lee Copeland in his book A Practitioner’s Guide to Software Test Design, is "At its core, testing is the process of comparing "what is" with "what ought to be."" [18].

In general, software testing is a process where the SUT is analyzed using a set of techniques based on the characteristics that are to be tested, and in which state the SUT is when the testing takes place. An example would be if a company is in the prototyping phase of developing a mobile-application. Then, it might be desirable to perform user testing where the usability of the application is evaluated to identify problems with the understandability or operability of the application.

Techniques

Software testing is often divided into three types of strategies: black box testing, white box testing and gray box testing. With the black box strategy, the testing is based on the descriptions of the SUT, e.g. the requirements and specifications. This implies that no knowledge of the actual implementation or design of the SUT is used, only the expected behavior. The white box strategy is the complement to the black box strategy. With white box testing the knowledge of the SUT is essential, meaning that the person performing the testing activity has to have some general programming skills in order to understand the implementation of the SUT. The combination of these two strategies resulted in the third strategy, the gray box strategy. Here the SUT is investigated to the point that the tester gets just enough understanding of how it is implemented in order to write more effective black box tests. The more they investigate the SUT, the closer to white box testing they get. [18]

Advantages of using black box testing is that the work of creating tests may be lightweight and done in parallel with the implementation of the system to test. This is because the tests are not dependent on the underlying code, but its functionality. This, on the other hand, results in a problem if the specification artifacts are not defined well enough, i.e. the functionality to test is not clearly stated. This is where the white box technique prevails, where the test cases are created by analyzing the system’s software. For white box techniques, the problem is that it is often time consuming and require the tester to have prior knowledge of the SUT to understand how it works. It may also be the case that the tester may be biased to write test cases based on the actual functionality of the SUT, instead of how it is supposed to function.
Levels of testing

Different levels of tests are associated to changes in a certain type of software artifact. These artifacts may be requirements and specifications, the source code or other design artifacts. For each software development activity, there is a level of testing associated to it [1]:

- **Acceptance Testing** - Determine whether the complete system fulfills the requirements given by the customer. Is done together with end users.

- **System Testing** - Determines whether the complete system functions with all components connected. A test of the whole system’s functionality, which assumes that underlying subsystems function individually.

- **Integration Testing** - Tests the functionality of interfaces between components and determines whether the communication is done correctly. Assumes that the interfacing subsystems function individually.

- **Module Testing** - The process of testing individual software modules, both in isolation and in interactions with other modules. A module is defined as a class in C++ or as a file in C.

- **Unit Testing** - The lowest level of testing where the functionality of the smallest software components is tested. These components may be functions, methods or even single mathematical expressions in the code.

The process of software development and the corresponding testing levels may be visualized by using the V-model [25], see Figure 2.1.

![V-model](image-url)

**Figure 2.1:** The V-model visualizing the software development levels
Needless to say that each testing level finds different faults, e.g. integration testing find faults in interfaces, and that depending on the development activity, the tester must choose the appropriate test activity. The horizontal lines in Figure 2.1 represents which test activity that answers for development in the different layers of the system. As an example, if a development activity takes place in a subsystem, it may be necessary to perform a new integration test. It may also be necessary to perform tests for the lower levels of the change as well, being module tests and unit tests.

**Regression testing**

Regression testing is a common technique used for maintenance of software. Regression testing is the activity of testing software that has been modified with the purpose to help ensure that the the modifications has not included any modifications to earlier functioning code. If the software does not possess the functionality it had before the modifications, the software has **regressed**. [1]

A regression test suite grows larger as the SUT grow as more functionality needs to be tested. It is the developer’s task to choose what test cases to use in the test suite, facing the major tradeoff: software coverage versus execution time. There are four techniques associated with regression testing [26]:

- **Retest all** - The simplest and most expensive technique where all the regression test cases are re-run.

- **Regression test selection** - A technique used to reduce the execution cost of the regression test session. A subset of the test suite is chosen with a set goal in mind. These goals can be focused on coverage of the SUT, minimization of test suite/removal of redundant test cases, and so called safety where every test case generating different output than the original SUT is chosen.

- **Test case prioritization** - A technique that prioritize the test suite, without removing any test cases, in order to improve the rate of fault detection. According to Duggal and Suri, there are 18 different prioritization techniques and several search algorithms aiming to find the optimal prioritization. Examples of such search algorithms are greedy algorithms, 2-optimal algorithms or genetic algorithms.

- **Hybrid approach** - A combination of regression test selection and test case prioritization, where a subset of the test suite is selected and prioritized to improve metrics such as APFD.

The regression test selection technique, simply known as test case selection, is the main focus of this thesis. Specifically the technique which consists of a selection based on the similarities of test cases. As mentioned before, de Oliveira Neto et al. applied a technique where the test cases were selected based on their textual similarities. The idea of this technique is to, based on the similarities/differences of test cases, to make a selection that maximizes the coverage of the SUT. Meaning that this technique aims firstly to remove redundancy of a test suite, and secondly to remove test cases that affects the coverage the least after all redundant test cases have been removed. Of the three similarity functions used by de Oliveira Neto et al., only the one called Jaccard Index is used in this thesis and is described in section 3.6 [9].

**2.3 Topic modelling**

Topic models are used in natural language modelling to analyze a collection of documents for determination of a set of abstract latent (hidden) topics that occur in the documents. By analyzing the linguistic data of the input documents, a topic model derives a set of topics based on the frequency of words as well as their co-existence with other words in the same
document. In other words, topic models are statistical models widely used in text mining applications and there are several variants of the topic model representation, where the one used in this project is the LDA-model that has been widely adopted in the text mining field [27]. Topic models are often described as "bag-of-words" models since they do not consider in which order the words appear in a sequence. For example the sequences "you’re a wizard, Harry" and "wizard, you’re a Harry" will be considered equal, even though the orders of the words are different. The "bag-of-words" name comes from seeing each sequence only as an unordered collection of words of fixed quantity.

![Figure 2.2: Topic model factorization of a corpus C where Θ is the distribution of documents over topics and φ is the distribution of topics over words.](image)

Practically, a topic model can be seen as a dimensionality reduction of a corpus where the expressed structure in the corpus is represented as a probability distribution of latent topics, which in term is a probability distribution over words. A graphical representation is seen in Figure 2.2 where the probability distribution of words over a corpus, denoted C, is factorized into the components φ and θ. The φ component represents the topic distribution over words, meaning that each topic has a set of words affiliated with them. The θ component represents the document distribution over topics, as in each document has a set of topics affiliated with them. In terms of probabilities, Figure 2.2 can be expressed as:

$$p(w|d) = p(w|z)p(z|d)$$

(2.1)

In other words, the probability distribution of words w within a document d, p(w|d), is the product of the probability distribution of words within a topic z, p(w|z), and the probability distribution of topics within a document p(z|d). In the LDA model, both of these factorized distributions are the result of drawing from a Dirichlet distribution. [28]

### 2.4 Dirichlet distribution

Dirichlet distribution is a multivariate probability distribution that describes K >= 2 variables θ₁, ..., θₖ. These variables together form a probability distribution called a (K-1)-simplex, described:

$$
\theta = \{\theta_1, \ldots, \theta_K; \sum_{i=1}^{K} \theta_i = 1, 0 \leq \theta_i \}
$$

(2.2)

In general, a simplex is a probability distribution of the outcome of a series of random variables, where the sum of all the possible outcomes equals 1 (100%). E.g. a scalar θ is a 0-simplex (point-simplex) if it has the value 1, since the sum has to equal 1. Equally, a 2-vector θ is a 1-simplex (line-simplex) if the elements are non-negative and their sum is equal to 1, and a 3-vector would be a 2-simplex (triangle-simplex) under the same conditions. Figure 2.3 shows the plane that is the 2-simplex.
The Dirichlet distribution, denoted as $\theta \sim \text{Dir}(\alpha)$, is parameterized by a $K$-vector of positive-valued parameters $\alpha = (\alpha_1, \ldots, \alpha_K)$, where $0 < \alpha_i < \infty$ for each $i$. The Dirichlet distribution is affiliated with Bayesian statistics and is often used in machine learning applications. In these areas, it is often used as a prior probability distribution, or prior for short. A prior is the probability distribution that expresses beliefs about a certain event before it has occurred. An example of a prior could be the probability distribution related to the final standings of the teams in the Swedish Hockey League in the end of the season 19/20, which is in the future from this report’s creation. In the terms of LDA there are two priors, one for the document-topic distribution and one for the topic-word distribution which will be further explained in section 2.5.

The probability density function of the Dirichlet is defined as:

$$p(\theta|\alpha) = \frac{\Gamma(\sum_{i=1}^{K} \alpha_i)}{\prod_{i=1}^{K} \Gamma(\alpha_i)} \prod_{i=1}^{K} \theta_i^{\alpha_i-1},$$

(2.3)

where the Gamma function, $\Gamma(\bullet)$, is a generalization of the factorial function (!) function for non-integer values [27]. The probability density function is visualized in Figure 2.4 which is the 2-simplex seen from the angle of its normal, along with the effects of some different values on the parameter $\alpha$.

The different values of $\alpha$ governs the shape of the resulting distribution $\theta$, both in terms of location of the ‘peaks’ and their strengths. Figure 2.4a displays the case where $\alpha = (1, 1, 1)$ where every outcome of the distribution is equally probable to occur. If then the values of $\alpha$ is decreased below 1, the concentration of the distribution moves toward the end points of the simplex, meaning that the outcome, $\theta$, is more probable to have one dimension being more dominant than the others than having an equal mixture of the dimensions. I.e. the distributions $(1, 0, 0), (0, 0.9, 0.1)$ and $(0.05, 0.15, 0.8)$ are more probable to occur than the distributions $(0.4, 0.5, 0.6)$ and $(0.33, 0.34, 0.33)$. This is visualized in Figure 2.4b where the corners of the triangle is more highlighted than the central part, here brighter colors imply stronger peaks. As $\alpha$ decreases further towards 0, the peaks gets thinner and comes closer to the shape of single lines, so called diracs, with the amplitude (strength) of $\frac{1}{3}$ while all the other values become closer to 0. When instead increasing the values of $\alpha$, the probability distribution becomes more concentrated on the center of the triangle, see Figure 2.4c. This implies that $\theta$ is more probable to contain a mixture of the dimensions rather than one dimension being more dominant than the others, the inverse of lowering the values below 0. As $\alpha$ is increased toward infinity, the peak in the center takes the shape of a dirac with amplitude 1.
2.4. Dirichlet distribution

The observant one might have noticed that to this point, only symmetrical distributions have been used (same values throughout \( \alpha \)). This is because the usage of a symmetrical prior indicates that the only prior knowledge of the distribution is the composition of how few/many dimensions it contains. Note that there may be far more dimensions than the 2-simplex discussed so far. If it were known in advance that the distribution was less probable to contain a certain dimension, one could lower the value of that dimension in the \( \alpha \) vector. The Dirichlet probability distribution would then be skewed as can be seen in Figure 2.4(d). The fact is that in this thesis, only symmetrical \( \alpha \) with values \( \leq 1 \) will be used, this is motivated further in section 3.5. If a symmetric distribution is desired, one can instead use a single scalar instead of a \( K \)-size vector as prior. This results in all the dimensions having the same concentration, hence the scalar is called the \textit{concentration parameter}.

**Multinomial distribution**

In Bayesian probability theory, the Dirichlet distribution acts as the \textit{conjugate prior} to the multinomial distribution. This implies that the two distributions come from the same distribution family, which in this case is the exponential distribution family, making the computations of a serial use of the two distributions easier. The multinomial distribution is similarly explained as the Dirichlet distribution where the input parameter is a \( K \)-vector of event probabilities where the sum of all probabilities equal 1, i.e. a \( (K-1) \)-simplex. The resulting random variables \( z_i \) indicate how many times each variable number \( i \) is independently observed. In conclusion the Dirichlet distribution takes a \( K \)-size vector, \( \alpha \), and gives a multinomial \( (K-1) \)-simplex, while the multinomial distribution takes a multinomial simplex-vector and gives the occurrence-vector \( z \). [27]
2.5 Latent dirichlet allocation

Latent dirichlet allocation (LDA) is a statistical model used to represent documents as a set of latent topics, introduced by Blei, Ng and Jordan in [27]. The idea behind LDA is to model documents as if they were generated from a set of topics, where a topic is a distribution over a vocabulary of words. It makes the assumption that each document is a mixture of a small set of topics and that each word in the document belongs to at least one of these topics. The LDA is a generative model which means that it can take a set of topics, with their associated words, and generate a document that would correspond to a mixture of these topics.

As an example, say that there are two topics, where one contains words affiliated with soccer and the other contain words affiliated with dogs. The LDA model then assumes that the process of generating a document of, say, 100 words about “dogs playing soccer” is to draw the words as a combination of the two topics. E.g. 30 words are randomly drawn from the dog-topic and 70 words from the soccer-topic.

Graphical model

A corpus is a collection of M documents, which is denoted as \( C = \{d_1, \ldots, d_M\} \). Each document is then described as a collection of N words, which is denoted as \( d = \{w_1, \ldots, w_N\} \).

![Graphical model representation of LDA using plate notation. Plate M denotes repetitions for each document in the corpus and plate N denote repetitions for the distribution of topics and words within each document.](image)

Figure 2.5: Graphical model representation of LDA using plate notation. Plate M denotes repetitions for each document in the corpus and plate N denote repetitions for the distribution of topics and words within each document.

Figure 2.5 displays the three-level hierarchical Bayesian model that is LDA. The parameters \( \alpha \) and \( \beta \) are parameters at the corpus-level, being sampled only once during the generation process. Parameter \( \alpha \) represents the prior knowledge of the topic distributions over the documents, and \( \beta \) equally represents the prior knowledge of the word distributions over topics. Note that \( \beta \) is sampled K times from a symmetrical Dirichlet distribution parameterized by a scalar, \( \eta \). This differ from the prior work made by Maskeri, Sarkar and Heafield, mentioned in section 2.1, where the \( \beta \) parameter is inputted by the user [21]. The \( \theta \) parameter is sampled at the document-level and the parameters \( z \) and \( w \) are sampled at the word-level. The \( w \) is the observed word that comes out of the process, being appended to the document in question. Using this hierarchical model, LDA assumes that the following generative process for the creation of the corpus with a vocabulary with word-ids, \( \{1, \ldots, V\} \), where \( V \) is the number of unique words in the corpus:

1. Parameters \( \alpha \) and \( \eta \) are derived from the user’s prior knowledge of the distributions
2. For each topic \( k \):
   - Draw a vector of word distributions over topics \( \beta_k \sim \text{Dir}_V(\eta) \)
For each document \( m \):

- Draw a vector of topic distributions over documents \( \theta_m \sim \text{Dir}(\alpha) \)
- For each word \( n \):
  - Draw a topic assignment \( z_{m,n} \sim \text{Multinomial}(\theta_m), z_{m,n} \in \{1, \ldots, K\} \)
  - Draw a word \( w_{m,n} \sim \text{Multinomial}(\beta_{z_{m,n}}), w_{m,n} \in \{1, \ldots, V\} \)

In conclusion, the generative process assumed by LDA begins to create a matrix of the probability of each word over each topic. This implies that each vector \( \beta_k \) is a \((V-1)\)-simplex, where every entry is the probability of the \( k \):th word belonging to that topic. Note that the Dirichlet distribution is for this step parameterized using a scalar \( \eta \), hence the probability distribution is equal for every possible outcome. For the Dirichlet distribution used to draw \( \theta_m \), on the other hand, the resulting vector is a \((K-1)\)-simplex where a vector is used as parameter, \( \alpha \). Using the distribution \( \theta_m \), a topic \( z_{m,n} \) is then drawn for each word to be generated in the document. The value of \( z_{m,n} \) will be an integer representing the id of the drawn topic, given the help of the multinomial distribution. Finally, by using the topic index \( z_{m,n} \), the word \( w_{m,n} \) from that topic is drawn using \( \beta \) as parameter to another multinomial distribution. [10]

### Inference of topics

Prior to this section, the LDA model has been used to describe the generation of a document with the latent information being known. On the other hand, to infer the latent information, being the topic distributions over documents, \( \theta \), topic assignments, \( z \), and word distributions over topics, \( \beta \), is the most computationally complex part. The problem is to approximate the model’s latent variables’ distribution based on the observed output, the documents. This is called posterior inference, since the problem is to infer the latent variables’ distribution after (post) the output has been observed. In summary, the problem is to compute the posterior distribution of the latent variables given the output document:

\[
p(\theta, z, \beta | w, \alpha, \eta) = \frac{p(\theta, z, \beta, w | \alpha, \eta)}{p(w | \alpha, \eta)}
\] (2.4)

The probability \( p(\theta, z, \beta | w, \alpha, \eta) \) denotes the probability of the latent variables \( \theta \), \( z \) and \( \beta \), given the known prior probability distributions \( \alpha \) and \( \eta \) and the known output corpus \( w \). For many Bayesian models, topic models included, the posterior is intractable to compute, hence approximations must be used instead [27]. Specifically, it is the denominator probability \( p(w | \alpha, \eta) \) that is intractable due to marginalization over the hidden variables, a complete explanation is given by Dickey in [29]. The approximation is usually based on Markov Chain Monte Carlo (MCMC) methods as in [30] or, as in this thesis project, Variational Bayes (VB) inference methods. The specific method used for the posterior inference is presented by Hoffman, Blei and Bach in [31]. The method is an online method, meaning that the posterior is updated with the arrival of new documents instead of re-reading the whole corpus as in batch-methods. For a detailed description of the inference algorithm, see the original paper or the source code available on GitHub[4].

### Instability of LDA

One of the major drawbacks of LDA is its non-deterministic behavior by the usage of probability distributions. This behavior causes the outcome of both the process of generating a new corpus and the inference of topics from an available corpus to be different between runs. One example where this becomes clear is when inferring topics from a corpus where the training documents are entered in different order between runs, the result being that the topics may

---

4https://github.com/blei-lab/onlineldavb
look completely different each re-run. This behavior is investigated by Agrawal, Fu and Menzies in [16] where the authors conclude that due to this topic instability, users of LDA should consider tuning the model parameters instead of using the default ones. The authors also present a tuning algorithm, which they call LDADE, based on differential evolution. They use this to automatically tune the $<k, \alpha, \beta>$ parameters, where $k$ is the number of topics to infer by the model. A version of the LDADE tuning algorithm is used in this thesis project to investigate the stability of the model, when used on this specific data set. The tuning process is further explained in section 3.5.
This chapter describes the methods used to reach conclusive answers to the research questions.

During this project, data from two different companies in the form of test cases is automatically extracted, modeled and selected with the help of the topic model LDA. This process is implemented using Python 3 and the open source library gensim which provide functionality for topic modelling and natural language processing [32]. The implementation is done with the black box strategy in mind, meaning that the only available knowledge of the SUT is the source code for the test cases. An algorithm for tuning the resulting topic model is also implemented, as it is made clear from the literature that the default parameters for creating the model often result in sub-optimal solutions [16].

3.1 Data sets

The data sets are provided from two separate companies, one active in the security and video surveillance industry, and the other active in the automotive industry. The data sets are completely independent and are complemented with coverage information regarding the characteristics of interest for respective data set. A summary of the two provided data sets can be seen in Table 3.1. The test cases are integration-level test cases, see Figure 2.1 meaning that they test the functionality of interfacing components in the SUT.

Note that this is all the data available from the companies and that there is no possibility to execute the test cases and gather dynamic information during run time. Due to Non-Disclosure Agreements (NDA), the company names as well as the given data sets will not be presented in the report.

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Surveillance Company</th>
<th>Automotive Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>Files</td>
<td>198</td>
<td>2118</td>
</tr>
<tr>
<td>Test Cases</td>
<td>1094</td>
<td>10409</td>
</tr>
<tr>
<td>Features</td>
<td>158</td>
<td>-</td>
</tr>
<tr>
<td>Dependencies</td>
<td>197</td>
<td>-</td>
</tr>
<tr>
<td>Test Steps</td>
<td>-</td>
<td>254</td>
</tr>
<tr>
<td>Total Execution Time</td>
<td>225 minutes</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 3.1: Summary of data sets from each company
3.2 Preprocessing

The two companies provide different coverage information for respective data set. The surveillance company provides a data set whose coverage is based on test features and test dependencies. A test feature is a system component that is validated with the completion of that specific test case. Examples of (very general) features could be "ZoomIn" and "ZoomOut". A test dependency is a test feature that needs to be tested and validated before executing said test case. It is important to understand that even while striving to maximize dependency coverage, no check is being performed for the validation of said dependency. The surveillance company’s data set also includes execution times for each test case. The automotive company, on the other hand, only provide coverage information of test steps which is described in natural language in the given test suite. I.e. a test step may be denoted as the string "@Test step Configure network" inside one of the test cases.

It is important to know that some of the numbers in the data sets differ from the ones used in the work by de Oliveira Neto et al. in [9]. The first difference is the number of test cases for the surveillance company, where they used 1096. The reason for this is that two of the original test cases had been refactored and a decision was made to exclude them from the data, thus only 1094 test cases were used. The second difference is the number of dependencies used for the surveillance company, where de Oliveira Neto et al. used 384 dependencies. As the test suite internally is divided in two parts, they distinguish between the dependencies for each part thus having 197 dependencies for each part. In this thesis, however, the unique dependencies are counted only once. The motivation for this is because that how the number of features is counted, and thus is reasonable to count only unique dependencies as well. The final differences are the number of test cases and the number of test steps for the automotive company. de Oliveira Neto et al. uses the numbers 1972 and 1093 for the number of test cases and test steps, respectively. These differences are due to modifications that have been made to the test suites between the studies. And since no version control system was used for the test repositories, the exact data sets used by de Oliveira Neto et al. are not available for this thesis.

3.2 Preprocessing

The extracted corpora contain words in the hundreds of thousands, where the majority of the words are non-distinguishing information for the topics to be created. Example of these words is very common words such as "the", "in" and "of" or words that occur in many of the input documents. Hence, the extracted corpora are filtered using a list of very common words. The Natural Language Toolkit (NLTK)\(^1\) is used as a static foundation of this list and is then extended using dynamic identification of words that occur often and in many test cases \([33]\). NLTK includes an algorithm for stripping the suffixes of words and transforming them into a stem. This algorithm is called the Porter Stemming Algorithm \([34]\). As an example, take the words "walk", "walked", "walking" and "walks". The algorithm would transform these four variants into the word "walk", which is the stem. When the corpora have been filtered and stemmed, they are used to create dictionaries, one for each corpus. The dictionaries are artifacts that contain the ids for each unique word in the corpus and acts as a translation between these. The preprocessing steps are visualized in Figure 3.1. The generated dictionaries are then used together with respective preprocessed corpus to create a bag-of-words (BoW) representation for each corpus. This representation is simply a word-count of the words that exist in a document. I.e. it represents a document as a list of id-count pairs, this is because integers are much easier to manipulate and compare than strings.

\(^1\)http://www.nltk.org/
3.3. Topic modelling

Figure 3.1: The work flow for creating dictionary and BoW representation

Finally a technique called *tf-idf feature selection* is used to identify the most relevant 10% of words for each BoW representation. This technique uses a metric based on term frequency and inverse document frequency (hence "tf-idf") score described as:

\[
tfidf(w, d) = \frac{w}{W} \log \frac{D}{d}
\]  

(3.1)

The parameter \( w \) represents the number of a term’s occurrences in a document and \( d \) represent the total number of documents it occur in. The parameters \( W \) and \( D \) represent the total amount of terms in a specific document and the total amount of documents respectively. In summary, a term occurring often and only in a few documents is given a high score, thus implying that the term is "relevant" to that document. [35]

These preprocessing steps are the same as in [16], since the tuning algorithm, described in [35] is similar as the one used in the article. The most obvious reason is that there is a lot of non-saying words contained in the documents, and that this extraction of relevant words result in the topics containing only words that are actually relevant to the corpora.

### 3.3 Topic modelling

In the process of creating the LDA model, the two essential artifacts are the training corpus as a BoW representation and the dictionary of unique words. The first artifact, the training corpus, is the result of the steps described in section 3.2 and is, as the name indicates, used to train the LDA model and to infer the latent topic distributions. The second artifact is used mainly for when visualizing results to the user, such as the word-topic distribution. The user also has to select the three parameters \( \alpha, \beta, \gamma \), which are respectively; the number of topics to infer, the topic-document distribution and the word-topic distribution.

Now, with the LDA model trained and topics identified, it can be used to transform an arbitrary input corpus to the LDA model’s vector-space representation. In our case the input corpus is the same used for training the model in the first place. This results in an LDA representation of the corpus containing information about which topics a document, a test case in our case, consist of. E.g. a test case may be consisting of 70% of topic 1 and 30% of topic 2. When indexing the LDA model with the corpus, the results are of the form of a 2D-matrix, where each row represents the topic distribution of a single document. I.e. row 1 contains a list of topic-percentage pairs for test case 1, such as [("topic1", 0.7), ("topic2", 0.2),
3.4 Test case selection

After the topic modelling is finished, the resulting LDA corpus is analyzed to create an NxN matrix consisting of the similarities between each test case, based on their topic distributions, where N is the total number of test cases. The matrix is made up by normalized values stating the percentage of similarity between test cases, 0 implying no similarity at all and 1 implying equality. Practically the matrix is symmetrical around the diagonal with the values on the diagonal being 1, since each test case is 100% similar to itself. This matrix is then used to select the final subset of test cases based on their similarities in topics covered. Starting from an empty selection, the test case with the lowest sum of total similarities is chosen. I.e. the test case that is most different from the others are chosen as the initial test case. The following test cases are then chosen based on how similar they are to the current selection, where the one least similar is chosen as the next test case. This goes on until the desired size of the selection has been reached, e.g. 70% of the initial test suite. This algorithm is a greedy heuristic, meaning that it only considers the "next best" test case. This implies that the selection algorithm risks ending up in local optimums instead of global optimums, but the algorithm is chosen based on the simplicity of the algorithm. For the gathering of characteristics coverage of the resulting test cases, the mean of 10 runs for each TCS technique is selected. I.e. 10 LDA models is created, for both default and tuned parameters, and 10 random selections is performed.

Comparison of results

For the comparison of the results of this thesis against the results by de Oliveira Neto et al. in [9], it is not possible to simply compare the coverages due to differences in characteristic extraction as explained in section 3.1. Instead the comparison is based on the differences of using either TCS technique compared to respective random selection used. I.e. the random selection used in each work is used as a base line, and the deviations of the other techniques compared to respective random selection in each measure point is the metric used for the comparison of techniques. An example is:

Take the measure point where 50% of the surveillance test suite remains. Then the default LDA model performs 4% better than the random selection used in this thesis, while the jaccard index technique used by de Oliveira Neto et al. performs 20% better than their random selection.

A complete comparison of the measure points is given in Appendix A. For simplicity the average performance for each technique is compared in the results, instead of comparing the performance in each individual point. This metric is only used for the surveillance company’s data set, since the differences in performance of the random selection of the automotive data set are to great.

3.5 Parameter tuning

The tuning process used to answer RQ2 is an adaption of the one presented by Agrawal, Fu and Menzies in [16], and is described in Listing 3.1 using Python syntax. The implementation is an LDADE algorithm aiming to minimize the so-called raw score, which is a metric that measures the stability of the LDA model created using the specific configuration. I.e. the tuning process aims to evolve a configuration that makes the LDA model more stable when the order of the training corpus is randomized. A stable LDA model refers to the concept of
3.5. Parameter tuning

being able to infer the same, or very similar, topics regardless of the order the training corpus.
The calculation of the raw scores is described in Listing 3.3. For the rest of this section, the
notation "[Number]" will be used to reference rows of code in the listings for explaining the
algorithms.

    def LDADE( npop=10, f=0.7, cr=0.3, it=3 ) :
        ...
        :param npop : Size of frontier ( population )
        :param f : Differential weight
        :param cr : Crossover probability
        :param it : Number of generations
        :return: The configuration with the best raw score
        ...
        pop = init_population( npop ) # Init 10 models with randomized configurations
        # Evolve population
        for evolution in range( 0, it ) :
            new_gen = []
            for i in range( 0, npop ) :
                tmp_conf = extrapolate( pop[i], pop, cr, f )
                if raw_score( tmp_conf ) > raw_score( pop[i] ) :
                    new_gen.append( tmp_conf )
                else :
                    new_gen.append( pop[i] )
            pop = new_gen
        return best_conf( pop )

Listing 3.1: Pseudo code for LDADE algorithm

The LDADE algorithm begins with an initialization of a population of 10 random config-
urations, $< k, \alpha, \eta >$, and setting them as the current generation [9]. The main part of the
algorithm is then to, for each configuration in the population, generate a new configuration
using an extrapolate function and compare the raw scores of the new and the old one [14-15],
keeping the one with best score. This is repeated 3 times using an outer loop, where each
iteration is called an evolution [11-19].

The extrapolate function is described in pseudo code in Listing 3.2. It is explained as first
selecting three random configurations from the existing population [9], then using a crossover
probability to decide if the old parameter should be kept or a new should be created [12]. A
new parameter is created as a combination of the three selected configurations and fitted
 to stay inside its respective value range [15]. This process is repeated for each of the three
parameters, resulting in a configuration where 0-3 parameters are modified.

    def extrapolate( old, pop, cr, f ) :
        ...
        :param old : The selected configuration to extrapolate
        :param pop : The initial population to use to extrapolate a new configuration
        :param cr : Crossover probability
        :param f : Differential weight
        :return: The new parameter configuration
        ...
        n1, n2, n3 = chose_random( pop, 3 )
        new_conf = [
            for i in range( 0, 3 ) : # For each parameter <k, alpha, eta>
                if cr < random( 0, 1 ) :
                    new_conf.append( old[i] )
                else :
                    new_fitted_conf = fit( i, (n1[i] + f*(n2[i] - n3[i]) )
                    new_conf.append( new_fitted_conf )
            return new_conf

Listing 3.2: Pseudo code for extrapolating algorithm
3.5. Parameter tuning

The pseudo code for the calculation of a configuration’s raw score can be seen in Listing 3.3. The algorithm consists of a 2-level for-loop which has the purpose to remove sampling bias. The inner loop creates 5 models with said configuration but with different ordering on the input corpus (test cases) [9-11]. The outer loop then calculates the overlap of the 5 models where their resulting topics are compared and a percentage is given on how different the sets are, value 1 implying completely different models [7-13]. A more detailed description of the topic model evaluation is given in section 3.6. The outer loop is repeated 10 times in order to avoid sampling bias. The final score is then selected to be the median value of these 10 repetitions [14].

```python
def raw_score(config):
    """:param config: The configuration which score is calculated
:return: The raw score of said configuration
""
    score = []
    for j in range(0, 10):  # Reduce sampling bias
        lda_models = []
        for i in range(0, 5):  # Create 5 models
            tmp_model = create_lda(config)
            lda_models.append(tmp_model)
        similarities = overlap(lda_models)
        score.append(median(similarities))
    return median(score)
```

Listing 3.3: Pseudo code for calculating similarity score between models (raw score)

Parameter selection

The parameters considered for this thesis project are explained in Table 3.2. The default parameters are due to the implementation of the LDA library used, gensim.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Default</th>
<th>Value range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$</td>
<td>100</td>
<td>[100, 250]</td>
<td>The number of topics to derive from the corpus</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>$1/k = 0.01$</td>
<td>(0, 1)</td>
<td>The prior (symmetrical) topic distribution over documents</td>
</tr>
<tr>
<td>$\eta$</td>
<td>$1/k = 0.01$</td>
<td>(0, 1)</td>
<td>The prior word distribution over topics</td>
</tr>
</tbody>
</table>

Table 3.2: Parameters tuned in this thesis project

Ideally, the number of topics to use for the LDA model would be equal to the characteristic of interest, e.g. the total number of features to cover. This would require the model to represent an accurate mapping between topics and features. However, due to the stochastic behavior of the LDA model, it will find topics that could only be approximated as the actual features. The value range is kept close to the actual number of features, although, the range is still kept quite large since there exist no notion of what would be a good number of topics to use for the representation.

In section 2.4 it was explained how the value range for $\alpha$ and $\eta$ was $(0, \infty)$, with 1 being the mid-point. Here they are instead kept within the "lower half", e.g. remember that $\alpha$-values $< 1$ implies that the documents become more likely to be represented by a small number of topics. This value range is selected since it is assumed that the individual test cases within each test suite only covers a few of the coverage characteristics each. It is also assumed that each topic is made up of a small number of words, hence the $\eta$-values $(0, 1]$. It is worth noting that only symmetrical values will be used for both the Dirichlet priors. Again, this is because there is no prior knowledge of which features that are more heavily tested or which terms
that belong to which topic. Relate to 2.4d where the prior is an asymmetric $\alpha$, resulting in a skewed probability distribution.

### 3.6 Topic model evaluation

This section describes the methodology used to evaluate the generated LDA models in order to answer RQ2.

#### Topic stability

The stabilities of the topics are heavily dependent on the number of most relevant words, $n$, to consider during the similarity calculations. This is best explained with an example:

Two topics, $t_1 = \{\text{snow}, \text{santa}, \text{communism}\}$ and $t_2 = \{\text{russia}, \text{snow}, \text{communism}\}$, are subjects of a topic comparison where the word order represents the most relevant words. Imagine $n = 1$, resulting in only the most relevant word for each topic being considered for the comparison, being $\{\text{snow}\}$ and $\{\text{russia}\}$ respectively. Using Jaccard index, the result would say that the topics are completely different. If instead $n = 2$, then the two most relevant words for each topic would be compared, being $\{\text{snow, santa}\}$ and $\{\text{russia, snow}\}$ respectively. The result would then imply that the topics are 50% similar, since the word “snow” occurs in both topics. Further, if $n = 3$, the results would indicate that the topics are 66% similar since “snow” and “communism” occur in both topics.

Agrawal, Fu and Menzies use the values $n = \{1, \ldots, 9\}$ for their analysis, meaning that up to 9 of the most relevant words for each topic is compared for similarity [16]. Because of that, their value range of $n$ is adapted in this thesis as well. An argument for not choosing larger values on $n$ is the computational complexity that comes with having larger data sets to compare. Miller argues that the number 7 is a reoccurring number everywhere you look, with examples as “the seven deadly sins”, “the seven seas” and “the seven daughters of Atlas in the Pleiades” [36].

#### Raw scores and overlaps

The raw score is the metric used to quantify the stability of a model, which is configured with a specific parameter set $< k, \alpha, \eta >$. As explained in the pseudo code of Listing 3.3, the raw score is calculated using an overlap-function that calculates the overlap between LDA models, who have been initialized with different corpus orderings. The overlap is calculated using a similarity metric called Jaccard index, which is the intersection over the union of two sets [19]:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}, 0 \leq J(A, B) \leq 1 \quad (3.2)$$

In other words, Jaccard index calculates the percentage of overlapping sets, as in the example. In our case, the sets are topics and the overlap is a metric of how large the common word-set is. The overlap-function takes a value, $n$, representing the number of most significant words for each topic to consider in the similarity calculations. The significance of a word is the probability of that specific word being drawn from the topic, where value 1 implies 100% probability to draw that specific word. With the use of Jaccard index, it is possible to represent the similarity of two LDA models as their distances between inferred topics. The distance is calculated as:

$$\text{Sim}(A, B) = 1 - J(A, B), 0 \leq \text{Sim}(A, B) \leq 1 \quad (3.3)$$
where a distance score of 0 implies no distance at all (exactly the same inferred topics). The other way round if the distance score is 1 between two topics, this implies that none of the $n$ most relevant words from the topics exists in the other. Note that the terms "similarity" and "distance" are used interchangeably throughout the thesis, both referring to the concept of a similarity-function being used.

The distances are then calculated for each pair of topics resulting in a matrix of size $K \times K$. See Figure 3.2 for a graphical representation for the calculation of raw scores. There, the blue square represents the distances between the pairs of topics, with values closer to 1 meaning the topics are more different. The minimum values for each topic is then selected, marked with red in the Figure 3.2. The minimum values represents each topic’s best match with another topic in the other LDA model. Note that this method may result in several topics from one model having the same topic from the other model as its best fit, meaning that there may not be a 1-1 relationship between the topics.

![Figure 3.2: Graphical model visualizing the calculation of the raw score](image)

The green cube in Figure 3.2 may be called the *sampling-median cube*, since it visualizes the process of taking the median value of the calculated similarities in order to arrive at the final raw score. The notation arises from the process of having a 3-level nested loop with main purpose to avoid sampling bias. In summary, a topic difference matrix is created once for each pair of LDA models, whereas a new set of LDA models is created for each sampling bias-loop (row [11-18] in Listing 3.3). In other words, the number of topic similarities calculated is equal to: ($\#sampling\_bias\_iterations \times \#pairs\_of\_LDA\_models \times \#topics^2$). In the default case, when using 10 sampling bias iterations, 5 models (giving 10 pairs) and 100 topics per model, a total of $10 \times 10 \times 100^2 = 1000000$ topic similarities is calculated. In the end, the final raw score can be expressed as taking the median three times out of the minimum values of the topic difference matrix:

$$\text{median}(\text{median}(\text{median}(\text{min}\_\text{values}(\text{topic}\_\text{difference}\_\text{matrix}))))$$

(3.4)

### 3.7 Visualization

The visualization of the results is divided into two parts: one that visualizes the stability of the LDA model when tuning is performed and one that visualizes the outcome of the similarity-
based test case selection. These methods will be used to visualize the results for both data sets.

Model tuning

The stability of the LDA models is visualized using two methods. The first method is a line chart visualizing the models’ change in raw score after having their configurations tuned. The x-axis represents the number of most relevant terms that are used for the similarity calculations, while the y-axis represents the calculated raw score. This method for visualization the stability scores is adapted from Agrawal, Fu and Menzies in their paper [16].

The differences in topic diversity is visualized using the Python library pyLDAvis[^1] which is a Python port of the original R-package implementation created by Sievert and Shirley [37]. The topic diversity is, using pyLDAvis, visualized using a 2-dimensional map of the topics. Each topic is there represented as a bubble, where bubble sizes represent the topics’ usage within the corpus. Finally, the distances between the bubbles represent their similarities, where bubbles close to each other represent topics with similar word sets. The diversity is then visualized by generating two of these topic maps with models created with different randomized order on the training corpus, having the rest of the configurations the same. I.e. there will be two topic maps prior to tuning and then two more with tuned configurations.

Test case selection

The visualization of the TCS will be the same as for de Oliveira Neto et al. in [9]. They visualize their results with line charts, in which the x-axis represent the percentage of the remaining test suite and the y-axis represent the percentage of covered characteristic. The results from this thesis project will be presented in a similar chart together with results from using random selection of test cases as a base-line. These line charts will be generated once prior to tuning and once with tuned configurations to visualize the affect of the tuning on the TCS.

This chapter presents the results of the project that act as the foundation to formulate the answers to the research questions.

### 4.1 Preprocessing

Table 4.1 shows the corpus sizes after applying the filtering process described in section 3.2. The results show that the surveillance company’s corpus is a larger data set to begin with than the automotive company’s. The size of the generated stoplist and the filtered corpus imply, however, that the surveillance company’s corpus contains a large amount of irrelevant words according to the tf-idf score. Almost 75% of the surveillance company’s corpus is filtered out, while only 40% was filtered out for the automotive company’s corpus.

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Surveillance Company</th>
<th>Automotive Company</th>
</tr>
</thead>
<tbody>
<tr>
<td>Words before filtering</td>
<td>862 763</td>
<td>596 964</td>
</tr>
<tr>
<td>Stoplist size</td>
<td>2027</td>
<td>191</td>
</tr>
<tr>
<td>Words after filtering</td>
<td>225 491</td>
<td>361 200</td>
</tr>
<tr>
<td>Dictionary size</td>
<td>4589</td>
<td>2191</td>
</tr>
</tbody>
</table>

Table 4.1: Results of corpus sizes before and after filtering is applied

### 4.2 Model stability

The methodology for tuning the LDA models, described in 3.5 is adapted and the resulting raw scores and topic maps are presented in this section.

**Topic stability**

The stability of the models is evaluated using the raw score metric described in 3.6. The raw scores are calculated for several values of \( n \), the number of most relevant words to consider in the similarity calculations. The change in stability is visualized with a graph displaying the *delta score*, which is the difference of the raw scores before and after tuning.
4.2. Model stability

Surveillance company

Figure 4.1 shows the calculated raw scores for the LDA model created with the test cases from the surveillance company as training corpus. The blue curve represents the raw scores for the LDA model configured with the default configuration, while the red curve represents the LDA model where the configuration is tuned based on the best score for that specific n-value.

![Figure 4.1: The raw scores for the default and tuned surveillance company models](image)

The difference in raw scores between the default model and the tuned model is seen as the delta score displayed in Figure 4.2. From these figures it can be seen that the models have the same score for when $n = 1$, but after that, the tuned model always has better scores. The largest difference in scores is for the cases $n = 2$ and $n = 4$.

![Figure 4.2: The delta score between the surveillance company models' raw scores](image)
4.2. Model stability

Automotive company

The raw scores displayed in Figure 4.3 show that the tuning of the model provided much better scores than for the model with default configuration. It can be seen that the tuned model only parted with value 0 when \( n \geq 7 \), while the model with default configuration have scores \( \geq 0.5 \) for every \( n > 1 \).

![Figure 4.3: The raw scores for the default and tuned automotive company models](image)

Figure 4.3: The raw scores for the default and tuned automotive company models

Figure 4.4 shows the delta scores for the models created for the automotive company. It can be seen that the delta score is equal to the raw score for the default configured model up to the case \( n \geq 7 \), which is expected since the tuned model is equal to 0 up to this point. The largest difference is for the case \( n = 7 \), while the smallest is for \( n = 1 \).

![Figure 4.4: The delta score between the automotive company models' raw scores](image)

Figure 4.4: The delta score between the automotive company models’ raw scores

Topic diversity

The topic diversity of the LDA models is visualized by a pair of topic maps, each pair representing one model being trained with different training corpus order. The spread of the
4.2. Model stability

topics shown in the maps is a representation of how close the topics are in terms of associated words. The size of the topics in the maps is a representation of their frequency within the corpus. The numbering of topics in the topic maps is trivial and only the distances and sizes are of relevance. For the topic maps where a tuned configuration is used, the selected configurations are the ones that gave the highest delta score. For the surveillance company, this means that two configurations are visualized since there were two configurations with the same delta score.

Figure 4.5: Topic diversity between two surveillance company models with different corpus order, using default configuration

Surveillance company

The diversity of the topics for the LDA model using the default configuration can be seen in Figure 4.5. The topic map shows that the LDA model configured with default parameters may result in topic distributions that are quite different between runs. The pair shows both a difference in frequency-sizes within the models as well as the distances between them. To the right in Figure 4.5, the model has inferred topics with little distance between them (seen in the fourth quadrant). To the left, on the other hand, the model has inferred topics that are a bit more different (seen in the second quadrant).

Figure 4.6 shows the topic map for the model using the tuned configuration for the case when $n = 4$, and where the number of topics is $k = 223$. This figure shows that the tuning resulted in the model being able to infer topics more consistently, both in distances between topics and their frequency-sizes. Note that the number of topics is more than twice the number used for the default configuration.

Figure 4.7 shows that the consistency of topic diversity is the same, even for lower number of topics. The configuration used for this topic map is the one used for case $n = 2$, where the number of topics is $k = 109$. This is similar to the number used in the default configuration.

One characteristic that all three topic maps posses is that there is a lot of overlapping between topics, especially in the tuned versions. This implies that there are few topics that have unique word sets.
4.2. Model stability

Figure 4.6: Topic diversity between two surveillance company models with different corpus order, using tuned configuration with $k = 223$

Figure 4.7: Topic diversity between two surveillance company models with different corpus order, using tuned configuration with $k = 109$

**Automotive company**

The topic map in Figure 4.8 shows the diversity of the inferred topics by the LDA model using the default configuration. It can be seen that the topic distribution is both irregular in terms of topic differences between each other as well as their frequency-sizes. However, it can also be
seen that there is not much overlapping between the topics, implying that the topics contain mostly unique sets of words.

Figure 4.8: Topic diversity between two automotive company models with different corpus order, using default configuration

Figure 4.9: Topic diversity between two automotive company models with different corpus order, using tuned configuration with $k = 237$

Figure 4.9 shows the topic map for the tuned LDA model of the automotive company corpus. The configuration used for this model is the one used for the case $n = 7$, where the
number of topics is \( k = 237 \). This configuration resulted in a topic distribution where the frequency-sizes seem to be consistent, however the diversity of topics vary a bit. I.e. the topic map on the right hand side has a slightly more diverse topic distribution than the left hand side. Finally it can be seen that the tuned model infer topics with more overlapping than for the default model.

### 4.3 Test case selection

The results of the TCS is visualized as graphs where a selection of test cases, being a subset of the complete test set, is plotted against the percentage of covered characteristic. Each graph displays the results from using the LDA models, both with default configuration and with tuned configuration, as well as using random TCS for reference. Each measure point is the average over 10 runs with the same configuration for the LDA models, and with different random seeds for the random selection.

![Figure 4.10: Coverage of features for selections of surveillance company test suite](image)

**Surveillance company**

The results in section 4.2 regarding the raw scores motivate the selection of the tuned configuration. The selected configuration is the one used for the case \( n = 4 \), which were one of the cases with the best delta score (which of the cases is trivial since they both gave very similar result regarding TCS).

The coverage of features is shown in Figure 4.10 where it can be seen that the use of the LDA model with default configuration provide a slightly better coverage than for the random selection. It is also shown that the coverage when using the LDA model with tuned configuration is worse than for both the default LDA model and the random selection, especially in the case when less than half of the test suite is kept.

In terms of covered dependencies, the random selection and the LDA model with default configuration are almost inseparable throughout all the measure points as can be seen in Figure 4.11. Similarly to the case of feature coverage, the tuned LDA model performs worse than the other techniques. This becomes clear yet again after half of the test suite has been excluded.

Figure 4.12 shows the graph of the total execution time of the selected subsets. It can be seen that the reference random selection is kept almost completely linear, which is to be
expected due to the average being taken of 10 runs where major deviations are flattened out. It can also be seen that the LDA model with default configuration provides a slightly higher execution time for half of the measure points and never provides a lower execution time than the random selection. On the other hand, the tuned LDA model provides lower execution times than the other two in almost every measure point. And yet again, the largest deviations can be seen after 50% of the test suite has been excluded.

Automotive company

For the test suite provided by the automotive company, test steps are used as the only coverage characteristic. The configuration used for the tuned LDA model is the one used for achieving the highest delta score, i.e. for the case where \( n = 7 \). The results of the selection of the test suite are shown in Figure 4.13. It is shown that the use of a tuned LDA model gives the lowest coverage for every data point, while the random selection provides the best coverage for almost every measure point. However, due to the default LDA model’s much higher
performance for small sizes of the test suite, it actually performs 1% better than the random selection on average. The tuned default LDA model, on the other hand, performs 11% worse than the random selection on average. The random selection provides nearly full coverage as far as to the point where half of the original test suite is excluded. The LDA model with default configuration provides nearly full coverage only up to the point when 30% of the test suite has been excluded. Only after the test suite has been reduced to 15% of its original size, the random selection is outperformed by the default LDA model.

Figure 4.13: Coverage of test steps of automotive company selection

Figure 4.14: Coverage of test steps of automotive company selection, when only selecting test cases containing test steps.

The coverage of the random selection indicates that there is a lot of redundancy among the test cases, and after further analysis this was confirmed. It was shown that only 1 314 of the 10 409 test cases actually contains coverage of test steps, hence it would be possible to remove over 85% of the test suite and provide full coverage. Another selection was then made, but this time only of these 1 314 test cases. Figure 4.14 shows the selection after performing a new tuning process and performing a selection using the three techniques. The figure shows,
4.3. Test case selection

Table 4.2: Performance of techniques compared to respective random selection for the surveillance company’s test suite, measured in %. Grouped in two for clarification of different random selections.

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Default</th>
<th>Tuned</th>
<th>JI</th>
<th>NL</th>
<th>NCD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Features:</td>
<td>103</td>
<td>90</td>
<td>134</td>
<td>135</td>
<td>55</td>
</tr>
<tr>
<td>Dependencies:</td>
<td>101</td>
<td>90</td>
<td>129</td>
<td>129</td>
<td>64</td>
</tr>
<tr>
<td>Execution time:</td>
<td>103</td>
<td>84</td>
<td>109</td>
<td>78</td>
<td>127</td>
</tr>
</tbody>
</table>

once again, that a high coverage is achievable using both the random selection and the default LDA model, indicating that there is still a lot of redundancy among the remaining test cases. Specifically, it can be seen that using the default LDA model it is possible to select only 5% of the original test suite and still cover approximately 60% of the test steps. And yet again it is shown that the tuned LDA model provides, by far, the worst coverage of the three techniques. It actually seems that the tuned LDA model chooses the exclude the test cases with the best coverage early in the selection process.

Comparison against earlier work

The prior work made on the data sets by de Oliveira Neto et al. in [9] presents three selections based on the similarity functions, Jaccard Index (JI), Normalized Levenshtein (NL) and Normalized Compression Distance (NCD). As described in section 3.4, the techniques used in this thesis and the ones used by de Oliveira Neto et al. are compared based on their performance against respective random selection. Table 4.2 shows the comparisons to respective random selection, where the scores indicate the coverage using each technique relative to the coverage using the random selection for the surveillance company’s test suite. It can be seen that in terms of covered features and dependencies, both of the LDA models perform worse than the techniques based on JI and NL. But the LDA models perform better than the technique based on NCD. In terms of execution time, however, the LDA models perform better than the techniques based on JI and NCD, and the tuned model is almost as good as the technique based on NL which provides the lowest execution time. In summary, the LDA models perform worse than the techniques based on JI and NL, but performs significantly better than the technique based on NCD.

Note that, as explained in section 3.1, the automotive company’s test suite used in this thesis differs a lot from the one used by de Oliveira Neto et al.. This makes it impossible to perform a fair comparison in the same way as for the surveillance company, since the random selections’ coverage of test steps differ to much. As an example, the default LDA model used in this thesis provides on average 7% better performance compared to the random selection, seen in Figure 4.14. If comparing the coverage achieved by the default LDA model and the random selection used by de Oliveira Neto et al., ignoring the differences in extracted test data, the LDA model performs on average 46% better than their random selection.
This chapter presents a discussion of the work that’s been made in terms of achieved results and the methods used.

5.1 Results

This section discusses the results springing from the usage of LDA models for representing the data sets from the two companies as well as the tuning of the models.

Model tuning

The results presented in section 4.2 provide support to the claim that the tuning process was a success. By using differential evolution, it was possible to evolve configurations that made the LDA models become more reliable in terms of topic stability and consistent topic diversity. The delta scores in Figure 4.2 and 4.4 show that the use of the LDADDE algorithm results in the models being more consistent in what topics they infer from the training corpora, meaning that they infer topics that look similar to their equivalent in a re-trained model. The diversity of the topics is also kept steady, as in the topics’ distances to each other are kept similar between runs as well as their spread, as is visualized in the topic map figures in section 4.2. However, the tuning for topic stability did not improve the coverage of the TCS, in fact the tuned LDA models provided the worst coverage in almost all cases.

Another problem identified with the tuning process is the time consumption for the process to finish. In our case, the tuning process took quite some time for both data sets, between 6-8 hours, which is the results of using a goal function metric designed to reduce sampling bias. As explained with the median cube in section 3.6, the calculation of one raw score requires 1000000 topic comparisons. In other words the goal function metric is quite complex, thus making it object for the discussion if the tuning process is worth the extra effort. The easy answer would be "no", the tuning process does not seem to be worth the effort. This is because the tuning process takes longer time than it would take to run the full test suite for the surveillance company, thus making it impractical for the TCS purpose. Also, as the resulting coverage graphs show, the improvement in stability of the models does not improve the TCS coverage for any of the two test suites’ characteristics which is a problem since the tuning needs to be done every time a new test case is added to the test suite. If instead the goal func-
5.1. Results

The evaluation process was aimed to improve coverage of characteristics, the evaluation process would not take as long since the number of comparisons would then be in the thousands instead of millions. One negative aspect would then instead be that, for the surveillance company, several tuning processes would be needed since the tuning would need be done for all three characteristics. But even though the tuning process would come up with configurations providing at least as good coverage results as when using default configurations, as is almost always the case according to the conclusion of Fu et al. in [38], the question remains if the time for tuning would exceed the gained time when lowering the size of the selected subset.

Test case selection

The use of the LDA model for representing the companies’ test suites did not have the sought after effects on the coverage of characteristics. The graphs in section 4.3 show that the usage of the LDA model were similar in performance to the random TCS used as reference. The LDA models with default configurations gave, on average, the best coverage in all cases, though only by little. The LDA models with tuned configurations were for both test suites, in fact, the worst of the three in most aspects.

The reason for this is believed to be the fact that the tuned models have more overlapping topics than the default models, as well as less spread of the topics. If relating to the topic diversity maps in section 4.2, the selection of test cases would go about to select test cases starting from the outer parts and move closer to the more concentrated parts. This means that as the test suites shrinks, the test cases that are removed first are the ones that cover topics in the more concentrated areas. If the LDA model consists of a lot of overlapping topics, this means that the underlying features/test steps of the SUT is modelled using topics looking very similar. The connection between topics and features may thus become more blurry and less like a 1-1 relationship, which is the optimal case. I.e. the tuned LDA models infer the test cases to be considerably more similar than the default LDA models, thus making the excluding of test cases harder since they all seem to test the same functionality based on their linguistic data. This can be seen especially in Figure 4.14, where the excluding of test cases for the tuned LDA model results in a drastic early decrease of coverage.

It should be remembered that the tuning process was made with the goal function focusing on creating a more stable model, no emphasis was made on the TCS during the tuning of the LDA models. It can thus be deduced that improving the LDA models’ ability to infer the same topics each re-run seems to lower the performance when using the model in TCS purposes.

Comparison to earlier work

In the previous work done by de Oliveira Neto et al., where the same test suites were used, it is shown that similarity-based TCS is applicable to both test suites with good results [9]. Table 4.2 shows that, for the surveillance company’s test suite, the LDA based selections is outperformed by the top two techniques presented by de Oliveira Neto et al. in their article, in terms of covered features and dependencies. When comparing the total execution times of the selected test suites, the comparison yields that none of the techniques, neither the ones based on LDA nor the similarity-based ones, affect time reduction significantly.

The random selection’s coverage of test steps for the automotive company, seen in Figure 4.13 indicated that there was excessive redundancy in the test suite as the technique yielded questionably high coverage. After further analysis of the test suite, it was shown that not even 15% of the extracted test cases contained coverage of test steps. Thus another selection was made, but this time with only the cases covering test steps. As discussed in section 5.1 the results from this selection provided slightly better coverage for the default LDA model but significantly worse for the tuned one. The random selection, however, still has a high coverage indicating that there is still a lot of redundancy, which differ from the random se-
5.2. Method

In this section, the methods and data sets used in this thesis are discussed and compared against other alternative methods.

LDA

The LDA model, presented by Blei et al. in [27] in 2003, is one of the more popular (if not the most popular) topic models used for information retrieval according to the found literature. This was the main reason for using the LDA model instead of any other topic model for the extraction of test cases. Another reason was that there seemed to be a lot of software libraries...
and packages with easy to use functions for manipulating the LDA model. Of these libraries the gensim Python library was selected due to its excellent documentation and tutorials of its usage. Since the creation of LDA was more than 15 years ago, there have been several implementations as well as modifications of the original model.

As explained in section 2.5, the LDA implementation used in this thesis uses variational bayes for inferring topics as in [31]. Another popular alternative is to use gibbs sampling as in the work of Thomas et al. in [13] and Porteous et al. in [30]. There have also been further development of alternative topic models other than LDA. One example is Latent Semantic Analysis which aims to infer the semantic relation between words occurring in the same paragraphs, see the work by Deerwester et al. in [39] and the more recent work by Hofmann in [40].

There have also been some researchers modifying the LDA model on a more conceptual level, where the interpretation of topic distributions is changed. One of these is the Correlated Topic Model, developed by Blei and Lafferty in [41], where they use a logistic normal distribution instead of the Dirichlet distribution for modelling of topic variability. It can be concluded from the literature, however, that of all the available topic models the LDA model is among the most popular. Among the found literature, it is often the case that LDA is the main model to use for topic modelling, and if it is not then it is at least mentioned to be an alternative. This can not be said for the rest of the existing topic models as they are mainly referred to in their original papers.

**LDADE**

The need for a stable model arise when wanting the model to provide similar results between runs, implying that the LDA models are created several times for the same test cases. This may be the case when developers regression tests within a continuous integration environment. In that case, each developer might run the test suite locally on their computer before merging their modifications to the common release-version of the SUT. It may also be the case that instead of having a static list of the available test cases the test cases are fetched dynamically from a base-folder on the computer, thus the input order of the test cases to the LDA model may differ depending on the developer’s folder setup.

Tuning of a machine learning algorithm is to change how the model learns, which in our case implies that we alter how the LDA model infer the topics from the corpus. However, even though tuning almost always improve a model it is seldom considered due to the complexity to implement or time it takes to perform the tuning. Fu et al. discuss this assumption in their article [38], but argues that tuning does not need to be time consuming or complex. The authors utilize the differential evolution (DE) optimizer, the same as in this thesis, in order to prove that it is possible to have a tuning process which is simple to implement and still effective. They conclude that by using the DE optimizer, it is sufficient to use as low as 50 to 80 evaluations in order to find tuning improvements. In the end it is the metric used by the goal function that determines the execution time of the tuning process, which leaves it up to the developer to choose a goal function that improves the model’s results as well as not to time consuming.

The method for tuning the LDA model is based on the LDADE algorithm developed by Agrawal, Fu and Menzies in [16]. They present a literature study where it is shown that the stability of topic models are seldom taken into consideration, and that users tend to use the default configurations. This was the main reason for investigating the correlation between stability in the LDA model and coverage achieved by using this model for TCS in this thesis project. A decision was thus made to keep the goal function of the tuning algorithm to focus on maximizing stability. When seeing the results for the coverage of the test suites, it would have been interesting to see how the tuning could affect the coverage instead of the stability. Unfortunately, there was not enough time for this since both the implementation and the execution of the LDADE algorithm took longer than anticipated. The authors of LDADE
also mention that they used a multi-core architecture together with a Scala implementation in order to improve the execution time. In this project, the Python library gensim was used which led to only using Python for the complete implementation, which is a programming language not optimized for parallel computations. One option could have been to use the Python multi-process library for some speed up on the execution time of the algorithm, or to use another programming language with support for parallel computations for calculating the similarities between several models at the same time.

Other metrics that could have been used for the evaluation of the LDA model are the coherence metrics and metrics focusing on evaluating the model’s ability to generate documents. Coherence measures the understandability and interpretability of topics, meaning how easy the topics are to understand for humans. These kind of evaluation metrics were, however, considered to be irrelevant in this thesis project because of the thought use case. No humans will try to interpret the topics inferred by the model, and the model will not be tasked with generating any new documents from the topics.

Data sets

As explained in section 3.1, there are some differences in the versions of data sets used in this thesis compared to the versions used in the work by de Oliveira Neto et al. in [9]. These circumstances made it harder to compare the results in a single graph showing each TCS techniques coverage. Instead it was decided to use a relative comparison, where the techniques were compared based on their performance relative to the random selection technique used as reference in the two studies. At least for the surveillance company’s data set, this was feasible, since the differences between the data sets were not that large, and the characteristics of the random selections were similar. For the automotive company’s data set however, the difference in number of extracted test cases was to large. Also, the coverage of the random selection used in this thesis for the automotive company’s test suite, seen in Figure 4.13, indicates that there is a lot of redundancy regarding the coverage of test steps. After further investigation it was found that only 1 314 of the test suite’s 10 409 test cases actually covered test steps, which resulted in a new selection being made upon this subset.

The data sets provided only contain information regarding the the two test suites, no information is provided of the actual functionality or design of the systems they test. Provided this information, the black box approach was the only alternative for the TCS. If instead, information would be provided regarding the SUT, such as source code or design specifications, it would be possible to also perform a dynamic analysis to identify e.g. sizes of sub-components within the systems. This implies that the gray or white box techniques could be used to append relative weights to the coverage characteristics in order to improve the fault detection ability of the test suites. E.g. if the largest component of a system is the video rendering component, then the test cases covering this component may be prioritized in order to cover a larger part of this sub-system, up to a point at least. However, this would imply additional complexity since the SUT would need to be analyzed before each selection and after every new modification being made to the system.

Reliability and validity

The results of this thesis come from analyzing data sets from two companies that are not open source and available to everyone. Hence, the results are limited to the used data sets and are also not validated by execution of the selected test suites unto the SUT. The time and coverage data is, however, exported from execution of the original test suites, so the results comes from up-to-date information regarding the test cases.

Majorly, the internal validity of this thesis is related to the implementation of techniques used for extracting and preprocessing the corpora as well as the implementation of the LDA model. Even though the method for extracting the corpora can not be presented due to the
5.3. The work in a wider context

NDA, the preprocessing steps and the implementation of the LDA model are described in this thesis as well as by the referenced authors.

Given that the LDA model is based on probability and that the order of the training corpus results in different inferred topics, this also induces sampling bias to the results. The LDADE algorithm is designed with this ordering bias in mind and performs repeated calculations in order to avoid bias, see the median cube in Figure 3.2. A remark on this section is that, in order to save time on the execution of the tuning process, one of these bias-removing calculations has been modified. When calculating the raw score, a fixed number of LDA models is trained using a randomized training corpus order. This fixed number has been decreased from 10 down to 5, resulting in comparison of 10 model-pairs instead of the original 45 model-pairs.

Source criticism

The majority of the sources used in this thesis are in the form of scientific journal articles or papers published in conference proceedings. Some books are also used to provide in depth explanations of common concepts of the related theory, such as the concepts of the LDA model and software testing. Almost every source were found using Google Scholar, the Linköping University Library or online databases accessible through Linköping University such as the ACM digital library or IEEE Xplore.

The sources have varying age where some are only a couple of years old, while others are older than 15 years. The younger articles are used as reference when discussing the relevance of this thesis as well as providing a foundation that this work built upon. The older articles are used as references for the related theory and to explain core concepts that have been used frequently in later work by others. The older sources are often cited by many and are created by people renowned for their work in respective area. E.g the article by Blei, Ng and Jordan from 2003 regarding the creation of the LDA model has been cited 26794 times according to Google Scholar [27]. The even older article by Porter in 1980 regarding the creation of the Porter Stemmer algorithm for text processing is cited 10400 according to Google Scholar.

The books used as sources all present both an introduction to as well as in depth of theories relevant for this thesis. One of the books is Introduction to software testing, by Ammann and Offutt [1], which is used to explain core concepts of software testing. Some of the sources are not complete books but selected chapters of books, as in the cases of [10] and [28].

5.3 The work in a wider context

The use of software testing is a means for improving the quality of a system as the functionality is verified and faults are found and fixed. But it is not only the testing process itself that improves the quality of the system. The design of the system is the most crucial process when it comes to develop testable components. Persson and Yilmazturk argue for the importance of the test planning and designing for test, as in designing testable components and providing coverage for test cases [42]. This is concluded by Kasurinen et al. as well in [43] where interviews with several organization managers show that quality is built in the development process rather than when testing. When it comes to TCS where the source code of the test cases is analyzed the importance of well written test cases, which are both effective and economic to perform and analyze, is made clear. Fewster presents four attributes that describe the quality of a test case [44]. First of all it should be effective in its ability to detect faults. It should also be economic in its execution, analysis and debugging, and it should be evolvable, as in how much maintenance that is needed whenever the SUT is modified. Lastly, a test case should be exemplary, meaning that it should reduce the number of test cases by testing more than one thing.

Test automation is believed to be a solution to the lack of available resources for testing, since the testing can then majorly be done without the help of a human tester. An observation made by Karhu et al. in an empirical study on software testing describes that test automation
alone is not enough for increasing coverage of test suites, but that the selection of test cases needs to be guided by humans [35]. They also observe, from interviews with employees of varying competence and from several companies in different areas, that rapid changes in the underlying infrastructure of a system hinder the use of test automation as the maintenance of the test automation system then also increases. By using standardized technological infrastructure, it benefits the maintenance and training costs for the automation tools. Another empirical study by Kasurinen et al. shows that some organizations actually avoid using test automation due to the believed additional costs and low investment value it offers [43]. They conclude that test automation should be considered as a tool for guarantying functionality of an already existing system, i.e. regression testing, but not as a tool for actively finding errors.
Conclusion

This thesis was done with the purpose of investigating the use of topic models to representing test suites that are to be subjects for TCS. Specifically, the LDA model was used to create a diverse selection based on the test cases’ topic distributions and the results were compared against those from previous work conducted on the same data set by de Oliveira Neto et al.. The results show that the use of the LDA model for the purpose of selecting a diverse selection is not as effective as other similarity-based approaches, such as NL, JI and NCD, and that tuning the model for stability made the TCS results worse than for default configurations.

An evaluation of the LDA model in terms of topic stability and topic diversity was also performed, where the model’s configuration were tuned to improve these characteristics. By using LDADSE, a tuning algorithm based on differential evolution, the model’s ability to infer consistent and evenly distributed topics was improved. However, the results also show that improvements in the LDA model’s stability does not imply improvements for its application in the TCS scenario. It is discussed that the quality of the LDA model depends heavily on the quality of the underlying linguistic data as well as the sizes of the individual test cases.

Further work could include an adaption of the implemented tuning algorithm, but with the goal function focusing to improving the coverage of test characteristics instead of model stability. It could also be investigated if any other tuning algorithm, such as a genetic algorithm, could be used instead of LDADSE, both to improve TCS results and the model stability even further. Future work could also include applications of the implemented LDA model unto other test suites where the test cases are expressed in natural language, in order to find out if the model could create a better representation for that suite. It would also be interesting to see how the model would perform when put to use in TCP, and use e.g. a goal function focusing on maximizing the APFD for the tuning process.
Bibliography


This appendix contains the full comparison of each technique versus the respective random selection. I.e. the techniques used in this thesis are compared against the random selection used in this thesis, and the techniques used by de Oliveira Neto et al. are compared against their version of random selection. The performances are measured as (coverage of selected technique / coverage of random selection), implying that a value of e.g. 104 indicate that the selected technique is 4% better than the random selection in that measure point. Figure A.1 shows the performance of the techniques investigated in this thesis project, while Figure A.2 shows the performance of the three techniques used by de Oliveira Neto et al. in [9].
<table>
<thead>
<tr>
<th>Test suite size (%)</th>
<th>100</th>
<th>95</th>
<th>90</th>
<th>85</th>
<th>80</th>
<th>75</th>
<th>70</th>
<th>65</th>
<th>60</th>
<th>55</th>
<th>50</th>
<th>45</th>
<th>40</th>
<th>35</th>
<th>30</th>
<th>25</th>
<th>20</th>
<th>15</th>
<th>10</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Features:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td>100</td>
<td>101</td>
<td>100</td>
<td>101</td>
<td>101</td>
<td>101</td>
<td>102</td>
<td>103</td>
<td>104</td>
<td>104</td>
<td>104</td>
<td>103</td>
<td>102</td>
<td>104</td>
<td>103</td>
<td>103</td>
<td>103</td>
<td>110</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Tuned</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>101</td>
<td>100</td>
<td>100</td>
<td>99</td>
<td>99</td>
<td>98</td>
<td>96</td>
<td>95</td>
<td>92</td>
<td>90</td>
<td>85</td>
<td>77</td>
<td>72</td>
<td>68</td>
<td>63</td>
<td>58</td>
<td></td>
</tr>
<tr>
<td><strong>Dependencies:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td>100</td>
<td>101</td>
<td>101</td>
<td>101</td>
<td>101</td>
<td>101</td>
<td>102</td>
<td>102</td>
<td>101</td>
<td>101</td>
<td>100</td>
<td>99</td>
<td>99</td>
<td>99</td>
<td>98</td>
<td>101</td>
<td>106</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tuned</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>101</td>
<td>100</td>
<td>99</td>
<td>98</td>
<td>96</td>
<td>95</td>
<td>91</td>
<td>88</td>
<td>83</td>
<td>76</td>
<td>71</td>
<td>68</td>
<td>63</td>
<td>62</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Execution time:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td>100</td>
<td>101</td>
<td>102</td>
<td>103</td>
<td>103</td>
<td>103</td>
<td>103</td>
<td>105</td>
<td>106</td>
<td>108</td>
<td>107</td>
<td>105</td>
<td>103</td>
<td>101</td>
<td>98</td>
<td>97</td>
<td>100</td>
<td>118</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tuned</td>
<td>100</td>
<td>101</td>
<td>100</td>
<td>101</td>
<td>99</td>
<td>97</td>
<td>95</td>
<td>95</td>
<td>94</td>
<td>93</td>
<td>89</td>
<td>85</td>
<td>82</td>
<td>72</td>
<td>64</td>
<td>63</td>
<td>61</td>
<td>65</td>
<td>67</td>
<td>66</td>
</tr>
</tbody>
</table>

Table A.1: Performance of the LDA model, with default and tuned configurations, compared to the random selection used in the thesis.
<table>
<thead>
<tr>
<th>Test suite size</th>
<th>100</th>
<th>95</th>
<th>90</th>
<th>85</th>
<th>80</th>
<th>75</th>
<th>70</th>
<th>65</th>
<th>60</th>
<th>55</th>
<th>50</th>
<th>45</th>
<th>40</th>
<th>35</th>
<th>30</th>
<th>25</th>
<th>20</th>
<th>15</th>
<th>10</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Features:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JI</td>
<td>100</td>
<td>103</td>
<td>105</td>
<td>109</td>
<td>114</td>
<td>126</td>
<td>128</td>
<td>129</td>
<td>140</td>
<td>147</td>
<td>157</td>
<td>163</td>
<td>169</td>
<td>164</td>
<td>162</td>
<td>158</td>
<td>141</td>
<td>125</td>
<td>106</td>
<td></td>
</tr>
<tr>
<td>NL</td>
<td>100</td>
<td>103</td>
<td>105</td>
<td>109</td>
<td>114</td>
<td>126</td>
<td>128</td>
<td>130</td>
<td>141</td>
<td>148</td>
<td>159</td>
<td>165</td>
<td>170</td>
<td>165</td>
<td>162</td>
<td>158</td>
<td>141</td>
<td>125</td>
<td>106</td>
<td></td>
</tr>
<tr>
<td>NCD</td>
<td>100</td>
<td>93</td>
<td>85</td>
<td>79</td>
<td>71</td>
<td>68</td>
<td>68</td>
<td>65</td>
<td>61</td>
<td>59</td>
<td>57</td>
<td>52</td>
<td>45</td>
<td>41</td>
<td>39</td>
<td>34</td>
<td>32</td>
<td>30</td>
<td>33</td>
<td>29</td>
</tr>
<tr>
<td><strong>Dependencies:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JI</td>
<td>100</td>
<td>101</td>
<td>103</td>
<td>105</td>
<td>108</td>
<td>108</td>
<td>117</td>
<td>118</td>
<td>114</td>
<td>123</td>
<td>120</td>
<td>128</td>
<td>131</td>
<td>135</td>
<td>146</td>
<td>150</td>
<td>170</td>
<td>208</td>
<td>155</td>
<td>117</td>
</tr>
<tr>
<td>NL</td>
<td>100</td>
<td>101</td>
<td>103</td>
<td>105</td>
<td>108</td>
<td>108</td>
<td>117</td>
<td>118</td>
<td>114</td>
<td>123</td>
<td>120</td>
<td>128</td>
<td>131</td>
<td>135</td>
<td>146</td>
<td>150</td>
<td>170</td>
<td>208</td>
<td>155</td>
<td>117</td>
</tr>
<tr>
<td>NCD</td>
<td>100</td>
<td>92</td>
<td>88</td>
<td>86</td>
<td>84</td>
<td>76</td>
<td>75</td>
<td>72</td>
<td>68</td>
<td>70</td>
<td>66</td>
<td>63</td>
<td>60</td>
<td>57</td>
<td>56</td>
<td>51</td>
<td>49</td>
<td>46</td>
<td>25</td>
<td>21</td>
</tr>
<tr>
<td><strong>Execution time:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JI</td>
<td>100</td>
<td>105</td>
<td>105</td>
<td>106</td>
<td>115</td>
<td>120</td>
<td>118</td>
<td>128</td>
<td>126</td>
<td>116</td>
<td>116</td>
<td>107</td>
<td>150</td>
<td>115</td>
<td>120</td>
<td>114</td>
<td>91</td>
<td>79</td>
<td>77</td>
<td>57</td>
</tr>
<tr>
<td>NL</td>
<td>100</td>
<td>95</td>
<td>93</td>
<td>85</td>
<td>88</td>
<td>93</td>
<td>94</td>
<td>100</td>
<td>96</td>
<td>92</td>
<td>84</td>
<td>59</td>
<td>76</td>
<td>58</td>
<td>49</td>
<td>54</td>
<td>60</td>
<td>71</td>
<td>75</td>
<td>63</td>
</tr>
<tr>
<td>NCD</td>
<td>100</td>
<td>102</td>
<td>101</td>
<td>106</td>
<td>113</td>
<td>116</td>
<td>113</td>
<td>120</td>
<td>108</td>
<td>108</td>
<td>118</td>
<td>93</td>
<td>124</td>
<td>107</td>
<td>107</td>
<td>116</td>
<td>137</td>
<td>210</td>
<td>185</td>
<td>222</td>
</tr>
</tbody>
</table>

Table A.2: Performance of the three similarity-based techniques compared to the random selection used by de Oliveira Neto et al.