UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
UNIDADE ACADÊMICA DE SISTEMAS E COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

VICTOR EDUARDO BORGES DE ARAÚJO

USING A THESAURUS IN TRACEABILITY RECOVERY BETWEEN BUG REPORTS AND TEST CASES

CAMPINA GRANDE - PB

2020

VICTOR EDUARDO BORGES DE ARAÚJO

USING A THESAURUS IN TRACEABILITY RECOVERY BETWEEN BUG REPORTS AND TEST CASES

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Professor Dr. Franklin de Souza Ramalho.

CAMPINA GRANDE - PB

2020

A663u Araújo, Victor Eduardo Borges de.
Using a thesaurus in traceability recovery between bug reports and test cases. / Victor Eduardo Borges de Araújo. – 2020.

10 f.

Orientador: Prof. Dr. Franklin de Souza Ramalho.
Trabalho de Conclusão de Curso - Artigo (Curso de Bacharelado em Ciência da Computação) - Universidade Federal de Campina Grande; Centro de Engenharia Elétrica e Informática.

1. Software artifacts. 2. Bug reports. 3. Test cases. 4. Traceability. 5. Thesaurus. 6. Bug reports. 7. Wordnet. 8. Coceptnet. 9. Information retrieval. 10. Machine learning. 11. Vocabulary unifier. 12. Vocabulary encountered I. Ramalho, Franklin de Souza. II. Título.

CDU:004(045)

Elaboração da Ficha Catalográfica:

Johnny Rodrigues Barbosa
Bibliotecário-Documentalista
CRB-15/626

VICTOR EDUARDO BORGES DE ARAÚJO

USING A THESAURUS IN TRACEABILITY RECOVERY BETWEEN BUG REPORTS AND TEST CASES

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA:

Professor Dr. Franklin de Souza Ramalho
Orientador – UASC/CEEI/UFCG

Professor Dr. Eanes Torres Pereira
Examinador – UASC/CEEI/UFCG

Professor Dr. Tiago Lima Massoni
Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 2020.

CAMPINA GRANDE - PB

Using a Thesaurus in Traceability Recovery Between Bug Reports and Test Cases

Victor Eduardo Borges de Araújo
Federal University of Campina Grande
Campina Grande, Brazil
victor.araujo@ccc.ufcg.edu.br

Franklin Ramalho
Federal University of Campina Grande
Campina Grande, Brazil
franklin@computacao.ufcg.edu.br

ABSTRACT

The software development process generates several textual artifacts that are mostly written in natural language. Establishing connections between these artifacts can have a positive impact on performing a variety of tasks, including code understanding and maintaining. The use of Information Retrieval (IR) and Machine Learning (ML) techniques in order to recover the traceability between bug reports and test cases has already been proposed, however, the results indicated the need for improvements, especially to deal with the differences in vocabulary. In this paper, we created a Vocabulary Unifier using a *thesaurus* to expand the vocabulary encountered in bug reports, aiming to unify their terms in line with the terms from the test cases. We evaluated the techniques comparing its recall, precision and f2-score rates with those reached by previous works, observing slight improvements in its values.

Keywords

software artifacts, bug reports, test cases, traceability, thesaurus, wordnet, conceptnet

1. INTRODUCTION

Most of the documentation that comes with a software system, such as requirements and design documents, bug reports, test cases, and system features, is written in natural language. Establishing traceability links between these artifacts can be of great help in a variety of tasks, such as code understanding, system maintaining, requirements tracking, impact analysis and code reuse [1].

In particular, the recovery of traceability between bug reports and test cases can help the software team in several tasks, such as classifying bug priority, bug fixing and analyzing the impact of implementing improvements in a system [2].

The use of Information Retrieval (IR) [3] and Machine Learning (ML) [4] techniques for recovering traceability between test cases and bug reports has still reached incipient results. [2] applied three IR techniques (Latent Semantic Index, Latent Dirichlet Allocation and Best Match 25) and one Deep Learning (DL) technique (Word Vector) in the traceability recovery process between the aforementioned artifacts. The obtained results were: an average recall below 40%, and an average precision below 30%. The author points out possible reasons for these results, such as the difference in terms used in the description of test cases compared to the terms used in bug reports. The quality of the writing of the bug reports can also be one of the impasses for the use of the techniques. For instance, when small and not precisely descriptive, the bug reports can obstruct (both for humans and for the semi-automatic techniques) the traceability of test cases.

Aiming to minimize these issues concerning differences in vocabulary, we propose in this work, an approach to unify the vocabularies through a *thesaurus* in the context of traceability recovery between bug reports and test cases extracted from Mozilla's QA Team. The *thesaurus* is created using either WordNet or ConceptNet to fetch synonyms for the words inside bug reports, for the purpose of expanding its terms, increasing the likelihood of the IR techniques to match a larger quantity of words with the terms from the target documents, the test cases. For evaluating the approach, we extracted its recall, precision and f2-score rates and compared them with those reached by the baseline work.

The task of recovering traceability links between bug reports and test cases remains an arduous and error-prone work. Minimizing the issues encountered in the previous works, aiming to get better results, could help in the automation of this task. With the *vocabulary unifier*, we expect to increase the recall values extracted from the techniques, once it would help recovering traces that were not recovered due to differences in vocabulary.

This paper is organized as follows. Section 2 introduces basic concepts to better comprehend this work. Section 3 presents the baseline study. Section 4 introduces our approach. Section 5 describes the experiment, whereas Section 6 discusses its results. Section 7 shows related works, whereas Section 8 states our conclusions.

2. BACKGROUND

2.1 Bug Reports

Bug reports allow users to inform developers of the problems they experience while using a software [5]. The main objective of a bug report is to offer details about a failure identified and then help developers investigate and fix the bug once its presence is confirmed [6]. Typically, a bug report contains some fields, such as *Bug Number*, *Summary*, *Platform*, *Component*, *Version*, *Creation Time*, *First Comment Text*, *First Comment Text Creation Time*, *Status*, *Product*, *Priority*, *Resolution*, *Severity* and *Is Confirmed*.

Table 2.1 illustrates an example of a bug report extracted from Mozilla's Bugzilla page[1], containing its ID, Title and Description. Bug report 1276120 describes an error related to autocomplete suggestions in the url bar of Mozilla's Firefox.

[1] https://bugzilla.mozilla.org/

ID	1276120
Title	Autocomplete suggestions just don't go away after Win+Down
Description	STR: 1. Open new window, switch it to normal (not maximized) mode 2. Focus urlbar, type "a" 3. Press Win+Down 4. Click button on Windows Taskbar to open the window again 5. (bonus) Focus searchbar, type "b". Move the window a bit (drag-n-drop) AR: After step 4 autocomplete suggestions are open. They don't hide/move in Step 5. ER: Either no autocomplete suggestions after step 4, or something more smart. This is regression from bug 1264983.

Table 2.1: Bug Report Example

2.2 Manual Test Cases

A manual test case (MTC) defines a textual sequence of steps to reproduce that should occur as described, each of which including the expected results, allowing a tester to determine if a software is following the stakeholder's requirements [7].

Table 2.2 shows a Mozilla Firefox's test case example linked to the bug report depicted in Table 2.1, describing steps to ensure that a star appears if the drop down displays a result that is marked as favorite in Mozilla's Firefox browser. It also expects that, when the favorite is clicked, its link will correctly open.

TC Number	16
TestDay	20160603 + 20160624 + 20161014
Generic Title	Awesome Bar Search
Ctr Nr	4
Title	Search State - Favorites
Steps	1. Ensure the star appears if the drop down displays a result that is marked as a favorite 2. Ensure clicking the link for the favorite opens the appropriate site
Expected Result	1. Search result has the star icon displayed 2. The link is correctly opened.

Table 2.2: Test Case Example

Mozilla Firefox's Test Case's template has the following fields: *TC Number*, *Test Day*, *Feature ID*, *Firefox Feature*, *Generic Title*, *Ctr Nr* (Control Number), *Title*, *Preconditions*, *Steps* and *Expected Result*.

2.3 WordNet

WordNet is an online lexical database designed for computational purposes. English nouns, verbs, adjectives, and adverbs are organized into sets of synonyms (synsets), representing a lexicalized concept [8].

In this paper, we use WordNet as a synonyms source in order to expand the artifacts vocabulary, as it provides *synonymy* relations, its basic relation that represents a symmetric relation between word forms.

WordNet also has a web application where a user can fully use its features. Figure 2.1, shows an partial example extracted from its web application for the term 'switch'. It brought out 14 synsets, divided into two categories, Nouns and Verbs, each one containing seven synsets, as can be partially seen in Figure.

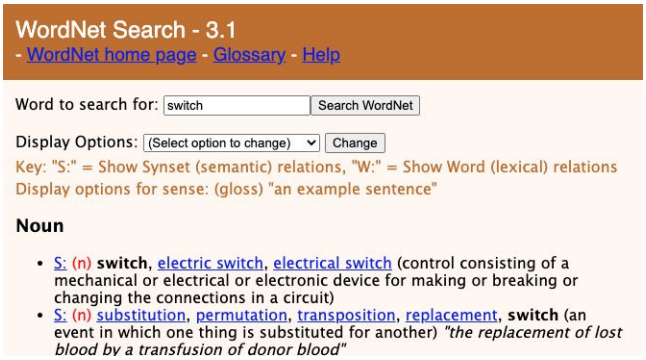

Figure 2.1: Example extracted from WordNet's web application[2]

Alongside with *synonyms*, *antonyms*, *hyponyms*, *meronyms* and other types of relations, Wordnet API also returns whether a term is described as a *noun* or not, a feature that we also use in this work.

2.4 ConceptNet

ConceptNet is a knowledge graph that connects words and phrases of natural language with labeled edges. Its knowledge is collected from many sources that include expert-created resources, crowd-sourcing, and games with a purpose. It is designed to represent the general knowledge involved in understanding language, improving natural language applications by allowing the application to better understand the meanings behind the words people use [9].

ConceptNet's feature set includes multi-language support as well as other types of relations than the ones from WordNet, such as *Related Terms*, *Derived Terms*, *Context of This Term*, *Ways of This Term*, etc. Although it has a wider range of features than WordNet, it should be emphasized that ConceptNet's list of sources includes WordNet.

Figure 2.2 displays a partial example of the features from ConceptNet taken from its web application. As we can observe in the image, for the term 'switch', it brings out a list of synonyms (in several languages), terms defined as *types of* 'switch' and *related terms*.

[2] http://wordnetweb.princeton.edu/perl/webwn

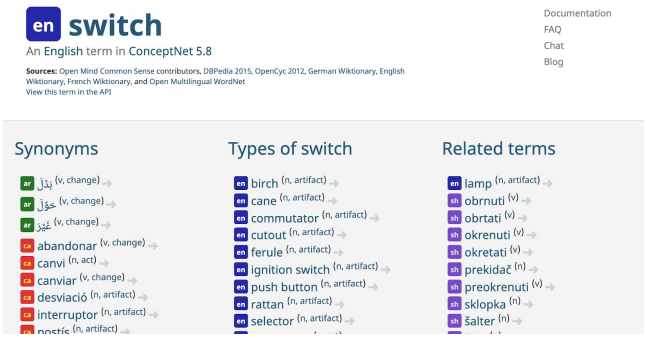

Figure 2.2: Example extracted from ConceptNet's web application[3]

3. BASELINE STUDY

[2] proposes an approach to recover traceability links between bug reports and test cases applying three Information Retrieval (LSI, LDA and BM25) techniques and one Deep Learning (Word Vector) technique. The data used in this study came from the Mozilla Firefox Quality Assurance Team. Aiming to enable a comparison between the IR and DL techniques, the same input and evaluation process was used for every technique, in which a comparison was made at the end with an oracle that is used as a reference for judging the effectiveness of the applied techniques.

The results encountered in the baseline study have shown that a couple of improvements could be made in order to increase its effectiveness and use the techniques in a semi-automatized traceability recovery process. The methodology from [2] provides a trustworthy way to compare recall, precision and f2-score metrics extracted from the techniques, enabling us to compare the impact of the introduction of the *vocabulary unifier* module.

3.1 Architecture

The baseline study consists in four steps: (i) the parse of the documents being traced, extracting the most relevant words; (ii) the creation of a *term-by-document* matrix that supports the traceability recovery made by a computational technique; (iii) the ranking of the most similar target documents by the computational technique based on the source document; and, at the end, (iv) the comparison between the rankings and the ground truth in order to get an evaluation of the technique.

Figure 3.1 extracted from [2] gives a general overview of the baseline study architecture. The Traces Builder module receives a set of bug reports and maps them to a subset of the provided test cases by applying (in isolation) each IR and DL technique. As a result, it is built a Recovered BR-TC Trace Links Matrix for each executed technique [2].

[3] http://conceptnet.io/

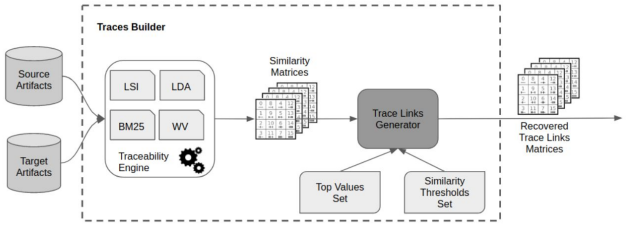

Figure 3.1: The General Overview of the Baseline Study Architecture extracted from [2] .

3.2 Results

The results obtained by the baseline study revealed the strengths and weaknesses of each applied technique in the task of traceability recovery between bug reports and test cases, showing that only one of them, LSI, has had satisfactory results, being feasible to be used in a semi-automatized traceability recovery process. Further details on the baseline results will be given in Section 6.

4. OUR APPROACH

In order to deal with the differences in vocabulary encountered between source and target documents during the traceability recovery process, we propose the use of a thesaurus, that we call *Vocabulary Unifier*, as a plugin module to the baseline study architecture.

The thesaurus plays the role of adding synonyms to the source document in order to unify the vocabulary used by the Mozilla Firefox QA Team and the testers. In our approach, we adopt WordNet as a synonyms source in order to expand our vocabulary, as it provides *synonymy* relations, its basic relation that represents a symmetric relation between word forms.We also adopt the ConceptNet as an alternative to WordNet when fetching synonyms for terms, as well as to check whether a term is scope related to the Computer Science/Programming field, as it provides an API that lists all the related terms to a given word.

As Figure 4.1 shows, the Vocabulary Unifier works as a plugin in the baseline study architecture, acting as an intermediary module between the input artifacts and the Traces Builder module. Therefore, the *Vocabulary Unifier* module receives the list of bug reports and test cases as input and delivers them to the traces builder module, with changes provided to the bug reports, by adding synonyms to the selected terms contained by them.

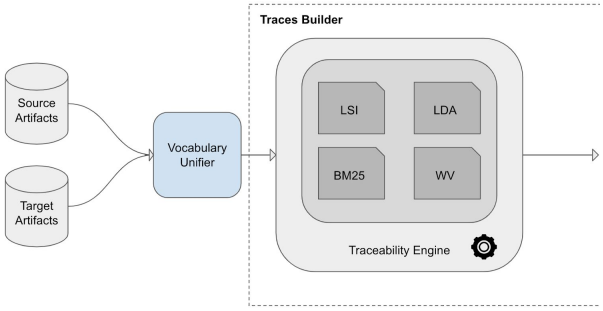

Figure 4.1: The partial architecture of the baseline research [2] with the addition of the Vocabulary Unifier plugin

4.1 The Vocabulary Unifier

The Vocabulary Unifier module receives two inputs: the source and the target documents. They then pass through a preprocessing phase where we tokenize them and remove stopwords that would be later also removed in the Traces Builder module from the baseline study. We chose to repeat that removal of stopwords to improve the algorithm's performance, given that it will handle considerably fewer words.

The Vocabulary Unifier module changes only the source documents (bug reports), once that the target documents (test cases) will be used strictly to create a corpus (on which the query will be applied) with their terms and frequencies.

In the expansion step, we iterate through the terms of the source documents and expand them according to the chosen API (either WordNet or ConceptNet) and the expansion techniques adopted. It is important to highlight that one synonym would only be added to the source document if it has a frequency above 0 on the created corpus. Therefore, the corpus created with the test cases are crucial, because it plays the role of a synonyms ranker, limiting the number of *synonyms per term* that will be added to the source documents.

We convert the tokenized expanded list of terms to a csv file, the format that is accepted as an input in the evaluation method from the baseline study. Notice that the only documents that were modified are the source ones. We used the target documents only to create the corpus on which the search is applied during the information retrieval task.

4.2 Expansion Techniques

In order to explore and investigate a greater number of scenarios in the vocabulary unifier, we adopt, in this work, eight expansion approaches to the terms, as follows:

- *WordNet Full Expansion (WFE)*: Every term is expanded using WordNet's API to fetch the synonyms;
- *ConceptNet Full Expansion (CFE)*: Every term is expanded using ConceptNet's API to fetch the synonyms;
- *WordNet Nouns Expansion (WNE)*: Only nouns are expanded using WordNet's API to fetch the synonyms;
- *ConceptNet Nouns Expansion (CNE)*: Only nouns are expanded using ConceptNet's API to fetch the synonyms;
- *WordNet Scope Expansion (WSE)*: Only technical (scope related) terms are expanded using WordNet's API to fetch the synonyms;
- *ConceptNet Scope Expansion (CSE)*: Only technical (scope related) terms are expanded using ConceptNet's API to fetch the synonyms;
- *WordNet Nouns and Scope Expansion (WNSE)*: Only technical (scope related) nouns are expanded using WordNet's API to fetch the synonyms;
- *ConceptNet Nouns and Scope Expansion (CNSE)*: Only technical (scope related) nouns are expanded using ConceptNet's API to fetch the synonyms.

To verify if a term is a noun, we used the provided API from WordNet that returns whether a term is a noun or not. To check if a term is a technical term (scope related), we fetch, from the ConceptNet API, the related terms to it and check if it has one of the following keywords: 'computing', 'computer programming', 'computation', 'computer', 'software', 'code', 'programming', 'programming language', 'program'. These keywords were selected by means of a manual process of searching for the most related terms of the scope.

5. EXPERIMENT

5.1 Scenarios

Considering the amount of parameters possibilities, we delve into eight expansion techniques besides the baseline study.

For each expansion technique, we explored a range from 1 to 5 *synonyms per term* in the expansion process. Therefore, we have a total of 5 scenarios for each expansion technique. We observed in a preliminary study that when we set this parameter with values higher than this range, the trend was the results to get worse.

For instance, consider the WNSE-2 from WNSE expansion technique. In this scenario, we use WordNet as the API to fetch synonyms, as well as we only expand one term if it is a noun and it is related to Programming scope.

Therefore, considering the term 'switch', extracted from the bug report 1276120 (shown in Table 2.1) the algorithm will evaluate it and check that: (i) it is a noun (at least in some scenarios); and (ii) it is related to the scope; In the sequel, it will fetch the synonyms from the WordNet API and receive a list of 22 synonyms for the 'switch' term. From these 22 synonyms, it will check which ones of them are on the corpus created with the frequency of the terms from the target documents. It will chop the 2 (by the WNSE-2 expansion approach) of these synonyms that appeared the most on the target documents. Therefore, for the term 'switch', the WNSE-2 approach adds the terms 'change' and 'alternate' (that appeared a total of 32 and 12 times, respectively, in the target documents) to the respective source document.

5.2 Evaluation

Our evaluation process is the same as the one used in the baseline study with a few key differences. First, we got rid of the DL technique, focusing on the IR ones.

However, we had to make a few adjustments in order to reduce the evaluation time, once we have a total of 45 scenarios . The main cut we did was the removal of the Word Vector techniques, once we focused on the IR techniques, alongside with the fact that it presented the lowest effectiveness in the baseline study.

Another change we made when evaluating our proposal was on the Top-N values parameters of the Traces Builder module. Instead of using the default 10, 20 and 40 values, we used 10, 20 and 120 values to amplify the impact that the proposed expansion approaches would have on the results, given that, the higher the Top-N value, the biggest the differences between the expanded scenarios and the baseline.

These adjustments made to the scope of this work are not mandatory for the vocabulary unifier to be executed. Our approach module can be used using the same parameters as the baseline study, including the DL technique and Top-40 instead of Top-120.

Even though the evaluation from the Traces Builder module brings out every metric used in the baseline study, we chose to focus on five of them when discussing the results: precision, recall, f2-score, the number of traces captured by every technique and the number of traces not captured by any of the

techniques. These metrics were chosen based on the main goal of using the *thesaurus*: increasing the recall values. Alongside with the recall metric, observing the number of traces captured by all and not captured traces will show us an in-depth look at the changes our approach made in a real problem. The precision and f2-score metrics will allow us to evaluate the impact of adding a thesaurus on the original base study beyond the recall metric and evaluate if the reached recall values are worthy.

The *precision*, *recall* and *f2-score* metrics, very common in the field of traceability recovery [10], they are defined as follows:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$\text{F2-Score} = \frac{5 \times Precision \times Recall}{4 \times Precision + Recall}$$

Where *TP* is the number of True Positives, *FP* is the number of False Positives and *FN* is the number of False Negatives.

6. RESULTS

In this section, we present the results extracted from the execution of each one of the cenarios, using mainly their average value in comparison to the baseline. We must point out that we also executed the baseline scenario without the Word Vector technique, using only the 3 IR techniques, once it would affect the recall, precision and f2-score final values.

When talking about expansion techniques, we will be aggregating its scenarios values in an average metric. It is important to point out that only the scenarios can be executed by the *vocabulary unifier* module. The expansion technique is just an entity to group out scenarios by its common parameters.

The results analysis were particularly complicated to be made. Although the differences between the values of the metrics were very relevant in several scenarios, tracking what was really decisive for these changes in metrics values was a challenge.

6.1 Traces Captured By All

When looking at the average number of traces captured by all techniques, as shown in Figure 6.1, four expansion techniques reached better values than the baseline: CSE, CNSE, WSE and WNSE. They all have in common the use of expansion, but only when the term has any relation with Computer Science/Programming (the scope conditional).

In fact, every scenario from CSE, CNSE, WSE and WNSE have higher values than the baseline scenario. In addition to them, the scenario WFE-1 also reached a higher number of traces captured by all techniques: 42.

TOP N	EXPANSION TECHNIQUE	TRACES CAPTURED BY ALL (avg.)
10	CSE	46.2
	CNSE	46.2
	WSE	45.4
	WNSE	45.4
	BASE	40
	CNE	36
	CFE	33
	WNE	28.8
	WFE	28.2

Figure 6.1: The average number of traces captured by all techniques for each expansion technique for Top-10

6.2 Not Captured Traces

Figure 6.2 depicts the number of not captured traces - by any of the techniques, applying 10 as the cut value for Top-N, where two expansion techniques reached equal or better values than the baseline using the average number of no captured traces.

TOP N	EXPANSION TECHNIQUE	TRACES CAPTURED BY ALL (avg.)
10	WNSE	212.8
	BASE	213
	WSE	213
	CNSE	217.6
	CSE	217.6
	CNE	235.2
	CFE	237.4
	WFE	278.4
	WNE	278.8

Figure 6.2: The average number of not captured traces for each expansion technique for Top-10

However, we must point out that WSE expansion technique has two scenarios, WSE-1 and WSE-2, that beat out the baseline, with 210 and 212, respectively, as the average number of not captured traces. In addition, the scenarios from WNSE that beat the baseline value are WNSE-1 and WNSE-2: 210 and 211, respectively.

When using the largest cut (Top-120), we can observe that 75% of the expansion techniques that use Vocabulary Unifier module have a lower average number of not captured traces than baseline scenario (that was also executed using the largest cut), as shown in Figure 6.3. This implies saying that 100% of the traces were found by at least one of the techniques in every CNSE, CSE, WNSE and WSE scenario using 120 as Top-N value. Each one of the 3 traces that were not captured by the baseline scenario under Top 120 value was captured using Vocabulary Unifier module with the scope conditional.

TOP N	EXPANSION TECHNIQUE	TRACES CAPTURED BY ALL (avg.)
120	CNSE	0
	CSE	0
	WNSE	0
	WSE	0
	CFE	0.2
	CNE	0.2
	BASE	3
	WFE	4
	WNE	4

Figure 6.3: The average number of not captured traces for each expansion technique for Top-120

6.3 Precision, Recall And F2-Score

Figure 6.4 shows the results for recall, precision and f2-score average values under BM25 technique.

MODEL	EXPANSION TECHNIQUE	RECALL (avg.)	PRECISION (avg.)	F2-SCORE (avg.)
BM25	WNSE	32,662	13,3	20,11
	WSE	32,616	13,388	20,144
	CNSE	32,312	13,114	19,868
	CSE	32,25	13,142	19,852
	BASE	32,16	13,06	19,64
	CNE	30,054	11,544	17,83
	WFE	29,074	10,416	16,932
	WNE	29,034	10,456	16,904
	CFE	28,866	10,672	16,864

Figure 6.4: The average values of Recall, Precision and F2-Score for each one of the expansion techniques using BM25 technique.

Using the BM25 technique, four expansion techniques reached better average recall values than the baseline: WNSE (the highest one), WSE, CNSE and CSE - all of them using the scope conditional as an expansion criteria. The highest value encountered from WNSE expansion technique was 32.79, using 1 as the *synonyms per term* value (scenario WNSE-1), an increase of 1.95% over the baseline recall value.

When looking at the average recall, precision and f2-score values for the LDA model, as shown in Figure 6.5, we also observe higher values in four expansion techniques, the same ones as using the BM25 model.

MODEL	EXPANSION TECHNIQUE	RECALL (avg.)	PRECISION (avg.)	F2-SCORE (avg.)
LDA	WSE	38,36	10,558	20,74
	WNSE	38,36	10,558	20,72
	CNSE	37,362	9,944	19,97
	CSE	37,09	9,846	19,77
	BASE	36,95	9,62	19,34
	CNE	33,886	8,324	17,262
	CFE	32,426	7,812	16,3
	WNE	32,376	8,574	16,706
	WFE	30,58	7,776	15,44

Figure 6.5: The average values of Recall, Precision and F2-Score for each one of the expansion techniques using LDA technique.

This time, WSE has the highest average recall value, having WSE-2 as its best scenario with a recall value of 38.77, increasing by 4.9% baseline scenario's value, the highest increase we encountered when looking at all of the three techniques.

In Figure 6.6 we can observe recall, precision and f2-score average values for each expansion technique under LSI technique. It is noteworthy that the LSI technique had the best results in the baseline study.

Once again, WNSE appeared as the best expansion technique, with its average value increasing recall baseline's value by 1.7%. The highest recall value was obtained by WNSE-1 scenario: 51.09, an increase of 3.4% over the baseline scenario's value.

MODEL	EXPANSION TECHNIQUE	RECALL (avg.)	PRECISION (avg.)	F2-SCORE (avg.)
LSI	WNSE	50,278	28,39	33,578
	WSE	50,25	28,382	33,548
	CNSE	49,808	28,238	33,292
	BASE	49,4	27,4	32,91
	CNE	48,264	27,396	31,836
	CFE	47,952	27,504	31,546
	CSE	47,408	28,238	33,292
	WFE	42,558	24,014	26,886
	WNE	41,514	23,734	26,238

Figure 6.6: The average values of Recall, Precision and F2-Score for each one of the expansion techniques using LSI technique

Every WSE, WNSE and CNSE scenario, in addition to the CSE-1 scenario, reached higher values than the baseline study under LSI technique.

The full set of results, including the execution of the techniques for every scenario (and the baseline scenario) can be found in a repository hosted on GitHub[4].

7. RELATED WORK

Santos *et al.* [11] also implemented improvements techniques to the research made by Gadelha *et al.* [2], such as text and information cleaning, spell-checking, terms weighting, similarity matrices merging and traceability matrices merging, achieving a recall of 93%. We can also cite a variety of previous studies that carried out comparisons between the effectiveness of IR and ML techniques in the traceability recovery context, such as Merten *et al.* [12] and Nilam Kaushit *et al.* [13].

As for the use of WordNet in the text retrieval context, Voorhees *et al.* [14] investigated the use of WordNet to enhance access to collections of text, exploiting its knowledge to ameliorate the effects synonyms and homographs have on text retrieval systems that use word matching. Loupy et al. [15] also used synonyms (as well as hyponyms) given by WordNet in order to enrich queries in the context of document retrieval.

Hsu et al. [16] compared the use of ConceptNet and WordNet in query expansion, which has been widely used to deal with paraphrase problems in information retrieval. In the conclusion, it showed that WordNet is good at proposing kernel words and ConceptNet is useful to find cooperative concepts, recommending the two resources as complementary tools to improve IR performance.

[4] https://github.com/victorrborges/thesaurus-traceability-study

8. CONCLUSIONS

This work evaluated the use of a *Vocabulary Unifier* when creating links between bug reports and test cases using IR techniques, focusing on recall, precision and f2-score metrics.

The main goal of this work was to investigate how a unifier vocabulary deals with IR techniques applied on the traceability recovery between bug reports and test cases. Although the results have shown that the use of a *thesaurus* made a slight impact over a baseline study recall values, we understand that, when applied with improvements made by other works (such as [11]), it could allow automatization when creating links between software artifacts or at least improve by a large margin the results obtained by the base study.

We believe that the Vocabulary Unifier can be used in other contexts than recovering traceability links between software artifacts. In fact, its use can be spread to any kind of textual artifacts, once there is a difference between the terms used in these artifacts (as well as differences in vocabulary, making it an issue). In cases where the textual artifacts do not belong to the Computer Science/Programming field, we would have to change the terms we used when verifying if a term belongs to the scope, using specific terms from the chosen field.

As future work, we suggest applying the Vocabulary Unifier module with the improvements made in [11]. We also recommend the study of other parameters to be used when applying the expansion further than the ones we used in this work (applying other expansion criterias) and the use of other APIs to fetch synonyms. Another path to follow is to modify the weight of the synonyms that were added to the source documents using the *thesaurus* when applying the IR techniques, in order to amplify or decrease their importance.

9. ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my research supervisor, Professor Dr. Franklin Ramalho (Federal University of Campina Grande), for providing me with invaluable guidance throughout this research.

I would also like to thank Professor Dr. Tiago Massoni (Federal University of Campina Grande) for the assistance in the construction of this research project.

I am extending my gratitude to my family and friends, for all the support they gave me during not only this research project but throughout my entire academic journey.

10. REFERENCES

[1] Antoniol, G., Canfora, G., Casazza, G., De Lucia, A., & Merlo, E. (2002). Recovering traceability links between code and documentation. IEEE transactions on software engineering, 28(10), 970-983.

[2] Guilherme Monteiro Gadelha. 2019. An Approach for Traceability Recovery between Bug Reports and Test Cases. Masters Dissertation in Computer Science - Federal University of Campina Grande (UFCG) (2019).

[3] Büttcher, S., Clarke, C. L., & Cormack, G. V. (2016). Information retrieval: Implementing and evaluating search engines. Mit Press.

[4] Alpaydin, E. (2020). Introduction to machine learning. MIT press.

[5] Nicolas Bettenburg, Sascha Just, Adrian Schröter, Cathrin Weiss, Rahul Premraj, and Thomas Zimmermann. 2008. What makes a good bug report? In Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering (SIGSOFT '08/FSE-16). Association for Computing Machinery, New York, NY, USA, 308–318. DOI:https://doi.org/10.1145/1453101.1453146.

[6] Dennis Lee. How to write a bug report that will make your engineers love you, 2016. Retrieved May 30, 2019 from https://testlio.com/blog/the-ideal-bug-report.

[7] Ian Sommerville. 2011. Software engineering 9th Edition. ISBN-10 137035152 (2011).

[8] George A. Miller (1995). WordNet: A Lexical Database for English. Communications of the ACM Vol. 38, No. 11: 39-41.

[9] Robyn Speer, Joshua Chin, and Catherine Havasi. 2017. "ConceptNet 5.5: An Open Multilingual Graph of General Knowledge." In proceedings of AAAI 31.

[10] Jane Huffman Hayes, Alex Dekhtyar, and Senthil Karthikeyan Sundaram. Advancing candidate link generation for requirements tracing: The study of methods. IEEE Transactions on Software Engineering, 32(1):4–19, 2006.

[11] Lucas Raniére Juvino Santos, Guilherme Gadelha, Franklin Ramalho and Tiago Massoni. 2020. Improving Traceability Recovery Between Bug Reports and Manual Test Cases. In the 34th Brazilian Symposium on Software Engineering (SBES '20), October 21–23, 2020, Natal, Brazil. DOI: https://doi.org/10.1145/3422392.3422424.

[12] Thorsten Merten, Daniel Krämer, Bastian Mager, Paul Schell, Simone Bürsner, and Barbara Paech. 2016. Do information retrieval algorithms for automated traceability perform effectively on issue tracking system data?. In International Working Conference on Requirements Engineering: Foundation for Software Quality. Springer, 45–62.

[13] Nilam Kaushik, Ladan Tahvildari, and Mark Moore. 2011. Reconstructing traceability between bugs and test cases: an experimental study. In 2011 18th Working Conference on Reverse Engineering. IEEE, 411–414

[14] Voorhees, E. M. (1998). Using WordNet for text retrieval. WordNet: an electronic lexical database, MIT press, 285-303.

[15] Loupy, C., & El-Bèze, M. (2002). Managing synonymy and polysemy in a document retrieval system using WordNet. In Proceedings of the LREC'02 Workshop on Using Semantics for Information Retrieval and Filtering.

[16] Hsu, M. H., Tsai, M. F., & Chen, H. H. (2006, October). Query expansion with conceptnet and wordnet: An intrinsic comparison. In Asia Information Retrieval Symposium (pp. 1-13). Springer, Berlin, Heidelberg.