Do Information Retrieval Algorithms for Automated Traceability Perform Effectively on Issue Tracking System Data?

Thorsten Merten[1], Daniel Krämer[1], Bastian Mager[1], Paul Schell[1], Simone Bürsner[1], and Barbara Paech[2]

[1] Bonn-Rhein-Sieg University of Applied Sciences, Dept. of Computer Science, Sankt Augustin, Germany
{thorsten.merten,simone.buersner}@h-brs.de,
{daniel.kraemer.2009w,bastian.mager.2010w,
paul.schell.2009w@informatik.h-brs.de}

[2] University of Heidelberg, Institute of Computer Science, Heidelberg, Germany
paech@informatik.uni-heidelberg.de

Abstract. [Context and motivation] Traces between issues in issue tracking systems connect bug reports to software features, they connect competing implementation ideas for a software feature or they identify duplicate issues. However, the trace quality is usually very low. To improve the trace quality between requirements, features, and bugs, information retrieval algorithms for automated trace retrieval can be employed. Prevailing research focusses on structured and well-formed documents, such as natural language requirement descriptions. In contrast, the information in issue tracking systems is often poorly structured and contains digressing discussions or noise, such as code snippets, stack traces, and links. Since noise has a negative impact on algorithms for automated trace retrieval, this paper asks: **[Question/Problem]** Do information retrieval algorithms for automated traceability perform effectively on issue tracking system data? **[Results]** This paper presents an extensive evaluation of the performance of five information retrieval algorithms. Furthermore, it investigates different preprocessing stages (e.g. stemming or differentiating code snippets from natural language) and evaluates how to take advantage of an issue's structure (e.g. title, description, and comments) to improve the results. The results show that algorithms perform poorly without considering the nature of issue tracking data, but can be improved by project-specific preprocessing and term weighting. **[Contribution]** Our results show how automated trace retrieval on issue tracking system data can be improved. Our manually created gold standard and an open-source implementation based on the OpenTrace platform can be used by other researchers to further pursue this topic.

Keywords: Issue Tracking Systems, Empirical Study, Traceability, Open-Source

1 Introduction

A considerable amount of requirements engineering (RE) research focusses on the automation of requirements traceability [10] by analyzing natural language (NL) of requirements artifacts (RA), e.g. [5,4,8]. These approaches report promising recall and accuracy. At the same time, RE research and best practices emphasize that high quality RAs should be written correctly, consistently, unambiguously, and organized[3]. In the context of automated trace retrieval, the performance of information retrieval (IR) algorithms benefits from documents that satisfy these criteria. However, the criteria are not satisfied by the data in issue tracking systems (ITS) [19]. Hence, the investigation and experiments in this paper are guided by the following main question:

Do information retrieval algorithms for automated traceability perform effectively on issue tracking system data?

To answer this question, a study with the data of the ITSs of four open-source projects is conducted. For each project a gold standard traceability matrix is created and the optimal results out of five IR algorithms with and without text preprocessing efforts, such as stemming and stop word removal, as well as the impact of term weights on different issue parts, are calculated and reported. The results show that algorithms perform poorly without considering the nature of ITS data, but can be improved by ITS-specific preprocessing and especially by term weighting. Furthermore, they show that VSM and LSI algorithms perform better than different versions of BM25 with ITS data.

The next section gives background information on ITSs and IR algorithms in the context of automated trace retrieval. It explains how traces are used and represented in ITSs exemplified by excerpts of our data and it gives a brief overview of related work in the field. Afterwards, Section 3 gives a brief overview of related work in the field. Section 4 states our research questions which are derived from the main question above. Section 5 explains the experiment setup including data acquisition, the employed tools, and algorithm evaluation. The, often counterintuitive, results are discussed in Section 6 for every RQ and it includes an overall discussion. Section 7 discusses how we mitigated threats to validity and finally, Section 8 concludes the paper and reflects on how future work can tackle the problem of ITS traceability.

2 Background

This Section briefly explains the employed IR algorithms (2.1) and how IR results are measured (2.2). Then, it introduces ITSs (2.3) and how data is handled in ITS. Finally, it bridges the gap between the nature of IR methods and the nature of ITS data (2.4).

2.1 Information Retrieval Background

IR algorithms are designed "to retrieve all the documents that are relevant to a user query while retrieving as few non-relevant documents as possible [2, p.4]". This definition can be applied to the problem of traceability: If I is the set of all issues, a relevance ranking of two issues i and $i' \in I$ can be computed by a function *similarity:* $I \times I \rightarrow R$,

[3] Among other criteria as defined in ISO/IEC/IEEE 29148:2011 [14].

with $R = \{r \in \mathbb{R} | 0 \leq r \leq 1\}$. Because a trace has only two states (it is either present or not), a threshold t is applied so that $\textit{trace}_t : I \times I \rightarrow \{true, false\}$ with

$$\textit{trace}_t(i, i') = \begin{cases} true & : \textit{similarity}(i, i') \geq t \\ false & : \textit{similarity}(i, i') < t \end{cases} \quad (1)$$

computes whether a trace between issue i and i' exists. A trace matrix of size $|I| \times |I|$ with elements $a_{i,j} = trace_t(i, j)$ can now be created.

The following IR algorithms were used to calculate the *similarity* function in our experiments: The vector space model (VSM) [27] using term frequency, inverse document frequency (TF-IDF), latent semantic indexing (LSI) [7] using the cosine measure, the Okapi best match 25 (BM25) [25] as well as its variants BM25L [16], and BM25+ [15]. The following paragraph gives a brief overview of the basics and differences of these algorithms. We refer the reader to IR literature for further information and details, e.g. [17,2].

VSM maps the terms of an issue to vectors. By using a distance metric such as TF-IDF, the similarity (S) of two issues can be computed. One of the main problems in VSM is exactly this dependency on each term and each term's spelling. Furthermore, the terms may have multiple meanings. Therefore, the VSM approach may compute a high similarity between issues with equal terms which may have different meanings due to context. LSI copes with this problem. Instead of computing S between terms, LSI computes S between concepts of the issues. Concepts are an abstraction of multiple terms and represent the "topics" of an issue. LSI creates those concepts using singular value decomposition [2, p. 101] which also reduces the search space. In contrast to the above, BM25 is a probabilistic approach to calculate S. It relies on the assumption that there is an ideal set of issues that are related to i and computes the probability of each issue to be in this set. BM25L and BM25+ both try to compensate problematic behavior of BM25 on long issues [16].
It is important to note that all of the approaches depend on the following properties of an issue i in the issue set I: a) the actual terms of i compared to another issue i', b) the number of terms (term frequency) in i, and c) the number of terms in i that are also in I (inverse document frequency). These properties can be influenced by text preprocessing. The most widely used preprocessing techniques are removing stop words (e.g. articles, prepositions, and conjunctions) and stemming (e.g. removing affixes; for example *connect* is the stem for connected, connecting, connection, ...)[4]. Due to these influences, it cannot be said which algorithm performs best with a certain data set without experimenting, although BM25 is often used as a baseline to evaluate the performance of new algorithms for classic IR applications such as search engines ([2, p. 107]).

2.2 Measuring IR Algorithm Performance for Trace Retrieval

IR algorithms for trace retrieval are typically evaluated using the recall (R) and precision (P) metrics with respect to a reference trace matrix. R measures the retrieved relevant links and P the correctly retrieved links:

[4] More preprocessing techniques are available. As an example [9] consider only nouns, adjectives, adverbs, and verbs for further processing.

$$R = \frac{\mathit{CorrectLinks} \cap \mathit{RetrievedLinks}}{\mathit{CorrectLinks}}, \qquad P = \frac{\mathit{CorrectLinks} \cap \mathit{RetrievedLinks}}{\mathit{RetrievedLinks}} \tag{2}$$

Since P and R are contradicting metrics (R can be maximized by retrieving all links, which results in low precision; P can be maximised by retrieving only one correct link, which results in low recall) the F_β-Measure as their harmonic mean is often employed in the area of traceability. In our experiments, we computed results for the F_1 measure, which balances P and R, as well as F_2, which emphasizes recall:

$$F_\beta = \frac{(1+\beta^2) \times \mathit{Precision} \times \mathit{Recall}}{(\beta^2 \times \mathit{Precision}) + \mathit{Recall}} \tag{3}$$

Huffman Hayes et al. [13] define *acceptable*, *good* and *excellent* P and R ranges. Table 3 extends their definition with according F_1 and F_2 ranges. The results section refers to these ranges.

2.3 Issue Tracking System Data Background

At some point in the software engineering (SE) life cycle, requirements are communicated to multiple roles, like project managers, software developers and, testers. Many software projects utilize an ITS to support this communication and to keep track of the corresponding tasks and changes [28]. Hence, requirement descriptions, development tasks, bug fixing, or refactoring tasks are collected in ITSs. This implies that the data in such systems is often uncategorized and comprises manifold topics [19].

The NL data in a single issue is usually divided in at least two fields: A title (or summary) and a description. Additionally, almost every ITS supports commenting on an issue. Title, description, and comments will be referred to as *ITS data fields* in the remainder of this paper. Issues usually describe new software requirements, bugs, or other development or test related tasks. Figure 1[5] shows an excerpt of the title and description data fields of two issues, that both request a new software feature for the Redmine project. It can be inferred from the text, that both issues refer to the same feature and give different solution proposals.

Links between issues are usually established by a simple domain-specific language. E.g. `#42` creates a trace to an issue with id 42. In some ITS the semantics of such traces can be specified (e.g. to distinguish duplicated from related issues). These semantically enriched links will be referred to as *trace types*. The issues in Figure 1 are marked as *related* issues by the Redmine developers. However, issues are also traced because of other reasons, including but not limited to:

– To express that a bug is related to a certain feature issue.
– To divide a (larger) issue in child-issues (e.g. for organizational purposes).

[5] Figure 1 intentionally omits other meta-data such as authoring information, date- and time-stamps, or the issue status, since it is not relevant for the remainder of this paper.

Fig. 1. Excerpts of two example issues from the Redmine Project

ITS data field	Issue #1910	Issue #12700
Title	Delete/close created forum entry	Let messages have a ”solved” flag
Description	I suggest a feature under the forums where the user can close or delete the topic he/she started. This way, other users will not get confused if the problem is already resolved.	It would be easier to go through the messages in the forums if there was a ”solved” flag users could set to show that their questions have been answered. * A filter could then be used to only show ”open” messages. [. . .]
Comments	[none]	[none]

In this paper, we report on the trace types *duplicate* and *generic*. A duplicate relation exists between two issues, if both issues describe exactly the same software feature and a *generic* relation exists, if two issues refer to the same software feature. This includes all the examples given above. Such a *generic* relation can for example be used to determine the total amount of time and money that was spent for a software feature, or to determine who was involved in developing, fixing, refactoring, and testing a feature.

Different semantics of an issue are subsequently referred to as *issue types*. ITS historically support the definition of one issue type per issue. Another approach is to tag issues with multiple descriptors[6]. In this paper we report on the issue types *bug* and *feature*, as well as the set of all issues that also includes uncategorized or untagged issues.

2.4 Impact of ITS data on IR algorithms

In previous research [19], we analyzed the content of NL in ITS data. We found that NL is often used imprecisely and contains flaws. Furthermore, NL is mixed with *noise* comprised of source code, stack traces, links, or repetitive information, like citations. Finally, the comments of an issue often drift from the original topic mentioned in the title and description towards something completely different (usually without being reorganized). Issues are seldom corrected and some issues or comments represent only hasty notes meant for a developer – often without forming a whole sentence. In contrast, RAs typically do not contain noise and NL is expected to be correct, consistent, and precise. Furthermore, structured RAs are subject to a specific quality assurance[7] and thus their structure and NL is much better than ITS data.

Since IR algorithms compute the text similarity between two documents, spelling errors and hastily written notes that leave out information, have a negative impact on the performance. In addition, the performance is influenced by source code which often contains the same terms repeatedly. Finally, stack traces often contain a considerable amount of the same terms (e.g. Java package names). Therefore, an algorithm might

[6] The researched projects use the ITSs Redmine and GitHub. In Redmine the issue type needs to be specified, GitHub allows tagging.

[7] Dag and Gervasi [20] surveyed automated approaches to improve the NL quality.

compute a high similarity between two issues that refer to different topics if they both contain a stack trace.

3 Related Work

Borg et al. conducted a systematic mapping of trace retrieval approaches [3]. Their paper shows that much work has been done in trace retrieval between RA, but only few studies use ITS data. Only one of the reviewed approaches in [3] uses the BM25 algorithm, but VSM and LSA are used extensively. This paper fills both gaps by comparing VSM, LSA, and three variants of BM25 on unstructured ITS data. [3] also reports on preprocessing methods saying that stop word removal and stemming are most often used. Our study focusses on the influence of ITS-specific preprocessing and ITS data field-specific term weighting beyond removing stop words and stemming. Gotel et al. [10] summarize the results of many approaches for automated trace retrieval in their roadmap paper. They recognize that results vary largely: "[some] methods retrieved almost all of the true links (in the 90% range for recall) and yet also retrieved many false positives (with precision in the low 10-20% range, with occasional exceptions)." We expect that the results in this paper will be worse, as we investigate in issues and not in structured RAs.

Due to space limitations, we cannot report on related work extensively and refer the reader to [3,10] for details. The experiments presented in this paper are restricted to standard IR text similarity methods. In the following, extended approaches are summarized that could also be applied to ITS data and/or combined with the contribution in this paper: Nguyen et al. [21] combine multiple properties, like the connection to a version control system to relate issues. Gervasi and Zowghi [8] use additional methods beyond text similarity with requirements and identify another affinity measure. Guo et al. [11] use an expert system to calculate traces automatically. The approach is very promising, but is not fully automated. Sultanov and Hayes [29] use reinforcement learning and improve the results compared to VSM. Niu and Mahmoud [22] use clustering to group links in high-quality and low-quality clusters respectively to improve accuracy. The low-quality clusters are filtered out. Comparing multiple techniques for trace retrieval, Oliveto et al. [23] found that no technique outperformed the others. They also combined LDA with other techniques which improved the result in many cases. Heck and Zaidman [12] also performed experiments with ITS data for duplicate detection with good recall rates. In addition they found that extensive stop word removal can be counter-beneficial for ITS data.

4 Research Questions

We divided our main question into the following four research questions (RQ):
RQ$_1$: How do IR algorithms for automated traceability perform on ITS data in comparison to related work on structured RAs? *We expect a) worse results to related work on RAs, due to little structure and much noise in ITS data, and b) BM25[+/L] variants to perform competitive for some projects.*
RQ$_2$: How do results vary, if ITS-specific preprocessing and weighting is applied? *We*

expect that removing noise improves results for all data sets as discussed in Section 2.
RQ$_3$: How do results vary for different trace and issue types? *E.g. [26,30,12] used IR algorithms on bug report duplicates, only. Since duplicates usually have a high similarity, we expect good results for duplicates.*
RQ$_4$: How do results vary between different projects? *Experiments are run with the data of four projects with distinct properties (see 5.1). We expect a wide range of results due to these differences.*

5 Experiment Setup

The experiment setup has three important steps: (1) The extraction and preparation of the data, (2) the manual creation of a gold standard traceability matrix to evaluate the experiment results, and (3) the automated trace retrieval by different algorithms, different preprocessing techniques, and different term weighting.

5.1 Data Preparation

Generally, 100 consecutive issues per project (in total 400 issues) were extracted from the respective ITS APIs. We focused on consecutive issues, since it is more likely that issues in such a set are related (e.g. because they refer to the same software features) [24]. Thus, the possibility to find meaningful traces is higher in a consecutive set of issues than in randomly selected samples.

The selection includes features, bugs, and uncategorized issues. The projects that rely on the Redmine ITS categorize more issues than the ones using the GitHub ITS[8] (see Table 1 for details). In addition, the extraction process followed existing links to other issues in a breadth-first search manner to make sure that the extracted dataset includes traces. Existing links were automatically parsed and collected into a traceability matrix (referred to as Developer Trace Matrix, DTM). Beside the NL data fields and the existing traces, meta-data such as authors, date- and time-stamps, the issue status, or issue IDs were extracted.

Researched Projects and Project Selection The data used for the experiments in this paper was taken from the following four projects:

- *c:geo*, an Android application to play a real world treasure hunting game.
- *Lighttpd*, a lightweight web server application.
- *Radiant*, a modular content management system.
- *Redmine*, an ITS.

The projects show different characteristics with respect to the software type, intended audience, programming languages, and ITS. Details of these characteristics are shown in Table 1. c:geo and Radiant use the GitHub ITS and Redmine and Lighttpd the Redmine ITS. Therefore, the issues of the first two projects are categorized by tagging, whereas every issue of the other projects is marked as a feature or a bug (see Table 1).

[8] This is discussed in depth in [19].

c:geo was chosen because it is an Android application and the ITS contains more consumer requests than the other projects. Lighttpd was chosen because it is a lightweight web server and the ITS contains more code snippets and noise than the other projects. Radiant was chosen because its issues are not categorized as feature or bug at all and it contains fewer issues than the other projects. Finally, Redmine was chosen because it is a very mature project and ITS usage is very structured compared to the other projects. Some of the researchers were already familiar with these projects, since we reported on ITS NL contents earlier [19].

Table 1. Project Characteristics

	c:geo	Lighttpd	Radiant	Redmine
Software Type	Android app	HTTP server	content mgmt. system	ITS
Audience	consumer	technician	consumer / developer	hoster / developer
Main programming lang.	Java	C	Ruby	Ruby
ITS	GitHub	Redmine	GitHub	Redmine
ITS Usage	ad-hoc	structured	ad-hoc	very structured
ITS size (in # of issues)	$\sim$ 3850	$\sim$ 2900	$\sim$ 320	$\sim$ 19.000
Open issues	$\sim$ 450	$\sim$ 500	$\sim$ 50	$\sim$ 4500
Closed issues	$\sim$ 3400	$\sim$ 2400	$\sim$ 270	$\sim$ 14.500
Sample size	$100 \approx 3\%$	$100 \approx 3\%$	$100 \approx 30\%$	$100 < 1\%$
Sampled issues with link	$\sim 50\%$	$\sim 20\%$	$\sim 12\%$	$\sim 70\%$
Issues labeled explicitly as **F**eature or **B**ug in sample	25F/26B	30F/70B	0F/0B	31F/61B
Project size (in LOC)	$\sim 130,000$	$\sim 41,000$	$\sim 33,000$	$\sim 150,000$

Gold Standard Trace Matrices The first, third, and fourth author created the gold standard trace matrices (GSTM). For this task, the title, description, and comments of each issue was manually compared to every other issue. Since 100 issues per project were extracted, this implies $\frac{100*100}{2} - 50 = 4950$ manual comparisons. To have semantically similar gold standards for each project, a code of conduct was developed that prescribed e.g. when a generic trace should be created (as defined in Section 2.3) or when an issue should be treated as duplicate (the description of both issues describes exactly the same bug or requirement). Since concentration quickly declines in such monotonous tasks, the comparisons were aided by a tool especially created for this purpose. It supports defining related and unrelated issues by simple keyboard shortcuts as well as saving and resuming the work. At large, a GSTM for one project was created in two and a half business days.

In general the GSTMs contain more traces than the DTMs (see Table 2). A manual analysis revealed that developers often missed (or simply did not want to create) traces or created relations between issues that are actually not related. The following examples indicate why GSTMs and DTMs differ: (1) Eight out of the 100 issues in the c:geo dataset were created automatically by a bot that manages translations for internationalization. Although these issues are related, they were not automatically marked as related. There is also a comment on how internationalization should be handled in issue (`#4950`). (2) Some traces in the Redmine based projects do not follow the correct syntax and are therefore missed by a parser. (3) Links are often vague and unconfirmed in developer traces. E.g. c:geo `#5063` says that the issue "could be related to #4978 [...] but I couldn't find a clear scenario to reproduce this". We also could not find evidence to mark these issues as related in the gold standard but a link was already placed by the developers. (4) Issue #5035 in c:geo contains a reference to #3550 to say that a bug occurred before the other bug was reported (the trace semantics in this case is: "occurred likely before"). There is, however, no semantics relation between the bugs, therefore we did not mark these issues as related in the gold standard. (5) The Radiant project simply did not employ many manual traces.

Table 2. Extracted Traces vs. Gold Standard

	Projects			
# of relations	c:geo	Lighttpd	Radiant	Redmine
DTM generic	59	11	8	60
GSTM generic	102	18	55	94
GSTM duplicates	2	3	-	5
overlapping	30	9	5	45

Table 3. Evaluation Measures Adapted from [13]

Acceptable	Good	Excellent
$0.6 \leq r < 0.7$	$0.7 \leq r < 0.8$	$r \geq 0.8$
$0.2 \leq p < 0.3$	$0.3 \leq p < 0.4$	$p \geq 0.4$
$0.2 \leq F_1 < 0.42$	$0.42 \leq F_1 < 0.53$	$F_1 \geq 0.53$
$0.43 \leq F_2 < 0.55$	$0.55 \leq F_2 < 0.66$	$F_2 \geq 0.66$

5.2 Tools

The experiments are implemented using the OpenTrace (OT) [1] framework. OT retrieves traces between NL RAs and includes means to evaluate results with respect to a reference matrix.

OT utilizes IR implementations from Apache Lucene[9] and it is implemented as an extension to the General Architecture for Text Engineering (GATE) framework [6]. GATE's features are used for basic text processing and pre-processing functionality in OT, e.g. to split text into tokens or for stemming. To make both frameworks deal with ITS data, some changes and enhancements were made to OT: (1) refactoring to make it compatible with the current GATE version (8.1), (2) enhancement to make it process ITS data fields with different term weights, and (3) development of a framework to configure OT automatically and to run experiments for multiple configurations. The changed source code is publicly available for download[10].

5.3 Algorithms and Settings

For the experiment, multiple term weighting schemes for the ITS data fields and different preprocessing methods are combined with the IR algorithms VSM, LSI, BM25, BM25+, BM25L. Beside stop word removal and stemming, which we will refer to as *standard preprocessing*, we employ *ITS-specific preprocessing*. For the ITS-specific preprocessing, noise (as defined in Section 2) was removed and the regions marked as

[9] `https://lucene.apache.org`

[10] `http://www2.inf.h-brs.de/~tmerte2m` – In addition to the source code, gold standards, extracted issues, and experiment results are also available for download.

code were extracted and separated from the NL. Therefore, term weights can be applied to each ITS data field and the code. Table 4 gives an overview of all preprocessing methods (right) and term weights as well as rationales for the chosen weighting schemes (left).

Table 4. Data Fields Weights (l), Algorithms and Preprocessing Settings (r)

Weight				Rationale / Hypothesis
Title	Description	Comments	Code	
1	1	1	1	Unaltered algorithm
1	1	1	0	– without considering code
1	1	0	0	– also without comments
2	1	1	1	Title more important
2	1	1	0	– without considering code
1	2	1	1	Description more important
1	1	1	2	Code more important
8	4	2	1	Most important information first
4	2	1	0	– without considering code
2	1	0	0	– also without comments

Algorithm	Settings
BM25	Pure, +, L
VSM	TF-IDF
LSI	*cos* measure

Preprocessing	Settings
Standard	
Stemming	on/off
Stop Word Removal	on/off
ITS-specific	
Noise Removal	on/off
Code Extraction	on/off

6 Results

We compute $trace_t$ with different thresholds t in order to maximize precision, recall, F_1 and F_2 measure. Results are presented as F_2 and F_1 measure in general. However, maximising recall is often desirable in practice, because it is simpler to remove wrong links manually than to find correct links manually. Therefore, R with corresponding precision is also discussed in many cases.

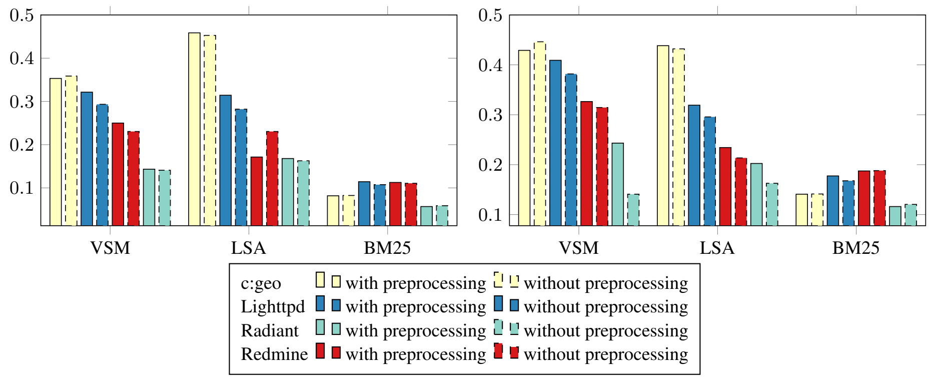
As stated in Section 5.1, a comparison with the GSTM results in more authentic and accurate measurements than a comparison with the DTM. It also yields better results: F_1 and F_2 both increase about 9% in average computed on the unprocessed data sets. A manual inspection revealed that this increase materializes due to the flaws in the DTM, especially because of missing traces. Therefore, the results in this paper are reported in comparison with the GSTM.

6.1 IR Algorithm Performance on ITS Data

Figure 2 shows an evaluation of all algorithms with respect to the GSTMs for all projects with and without *standard preprocessing*. The differences per project are significant with 30% for F_1 and 27% for F_2. It can be seen that standard preprocessing does not have a clear positive impact on the results. Although, if only slightly, a negative impact on some of the project/algorithm combinations is noticeable. On a side note, our experiment supports the claim of [12], that removing stop-words is not always beneficial on ITS data: We experimented with different stop word lists and found that a small list that essentially removes only pronouns works best.

In terms of algorithms, to our surprise, no variant of BM25 competed for the best results. The best F_2 measures of all BM25 variants varied from 0.09 to 0.19 over all projects, independently of standard preprocessing. When maximizing R to 1, P does not cross a 2% barrier for any algorithm. Even for $R \geq 0.9$, P is still < 0.05. All in all, the results are not good according to Table 3, independently of standard preprocessing, and they cannot compete with related work on structured RAs.

Fig. 2. Best F_1 (left) and F_2 (right) Scores for Every Algorithm

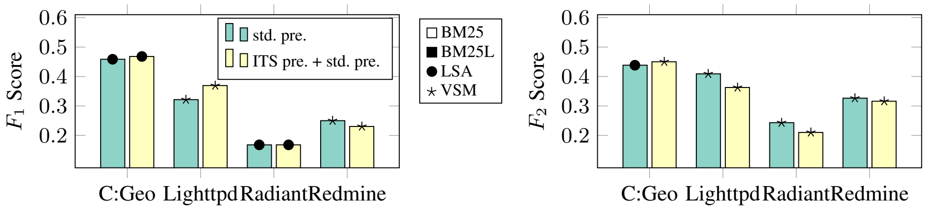


Although results decrease slightly in a few cases, the negative impact is negligible. Therefore, the remaining measurements are reported with the standard preprocessing techniques enabled[11].

6.2 Influence of ITS-specific Preprocessing and Weighting

This RQ investigates in the influence of *ITS-specific preprocessing* [12] and *ITS data field-specific term weighting* in contrast to *standard preprocessing*.

Fig. 3. Best Results With and Without Removing Noise

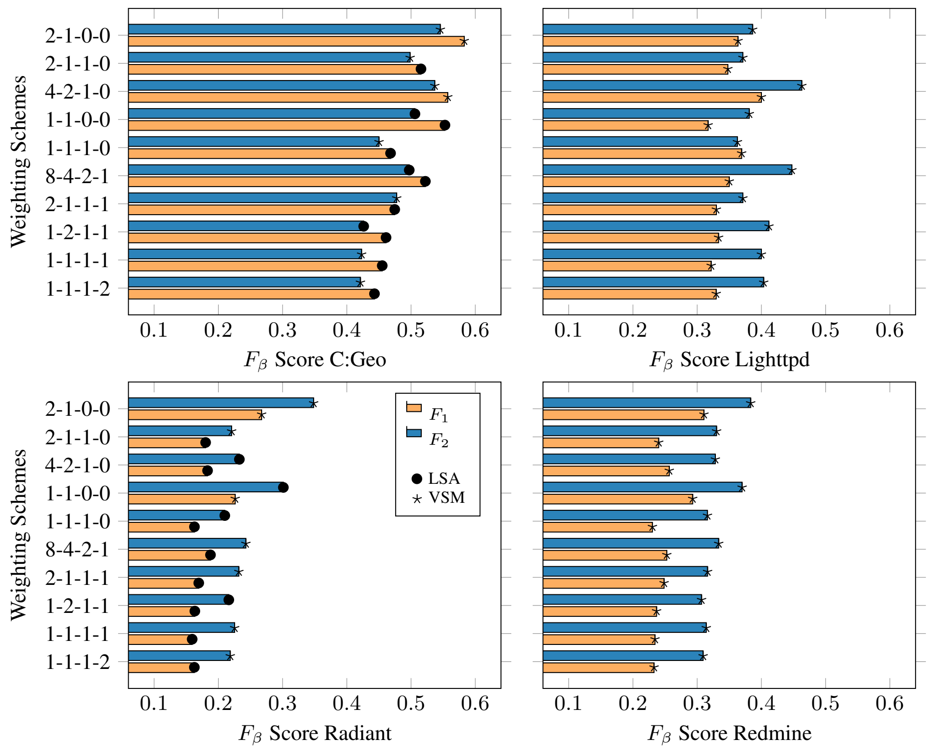


[11] In addition, removing stop words and stemming is considered IR best practices, e.g. [17,2].
[12] Removing code snippets and other noise can be achieved automatically, e.g. [18].

Contrary to our expectations, ITS-specific preprocessing impacts only c:geo clearly positively as shown in Figure 3. For the other projects, a positive impact is achieved in terms of F_1 measure only. Since preprocessing always removes data, it can have a negative impact on recall. This is what we notice as a slight decrease of the F_2 measure for three of the projects (4% for Lighttpd, 2% for Radiant, and 1% for Redmine). Overall however, precision improves with ITS-specific preprocessing.

Figure 4 shows the influence of different term weights in each of the projects. For a better comparison, the results are shown with *standard* and *ITS-specific preprocessing* enabled. The left axis represents the term weight factors for: *Title - Description - Comments - Code*. In contrast to ITS-specific preprocessing, Figure 4 shows that some term weights clearly performed best. In general, the weighting schemes that stress the title yielded better results. In addition, the figure also shows that code should not be considered by IR algorithms for trace retrieval: Term weights of 0 for code yielded the best results.

Fig. 4. Influence of Term Weighting



6.3 Influence of Trace Types and Issue Types

Issue Types Table 5 shows the best achievable results for F_1, F_2 and R on fully preprocessed datasets. The best results per issue type are printed in bold font. Since the Radiant dataset does not provide information on issue types, it is excluded in Table 5.

Trace retrieval from feature to bug issues worked best for the Lighttpd dataset. For Redmine retrieval between features worked best and for c:geo retrieval between bugs worked best; here, however, retrieval for other cases is much lower. Interestingly, there was no issue type, that worked best or worst for all projects.

Trace Types Table 6 compares the best achievable results for $\textit{trace}_t : I \times I$ and $\textit{trace}_t^{duplicate} : I \times I$. We restricted the comparison to generic relations and duplicates, since other annotated trace types in the GSTM[13] left too much room for interpretation by the annotators. E.g. it is hard to define when exactly an issue “blocks” another issue, without detailed knowledge of the project.

Table 6 shows that duplicate issues are detected competitively for c:geo and Redmine and rather poorly for Lighttpd. The latter contradicts our expectations for this RQ. However, a manual inspection of the data showed that duplicated issues often use different words to express the same matter, similar to the example given in Figure 1. This can only be resolved by domain knowledge and/or knowledge of domain-dependant synonyms. Both of which cannot be handled by standard IR algorithms without additional effort. Note, that we cannot report on the Radiant dataset, since the GSTM does not contain any duplicates as shown in Table 2.

6.4 Results per Project and Overall Discussion

Table 7 summarizes the best results per project for $F_{1/2}$ and R with (P) as well as the necessary settings to achieve these results. A baseline is represented by the best performing algorithm with *standard preprocessing* but without *ITS-specific preprocessing* and without *ITS data field-specific term weighting*. Although all results exceed this baseline, the positive impact of the ITS-specific efforts is only significant for c:geo and Radiant datasets ($F_{1,2}$ increase between 10 and 12%) and it has only a small impact on the Lighttpd and Redmine datasets ($F_{1,2}$ increase between 5 and 8%). This correlates with the ITSs that the projects employ. We hypothesize that data cleanup and weighting have a higher influence on the Github based projects, since the NL data looks a bit *untidy* in comparison to the Redmine based projects. With an improvement of 11% for F_2 the best values were achieved for c:geo and Radiant. We think that this is because both ITSs contain the least technical discussions and terms. On the contrary, the next best results are measured for Lighttpd and the project’s ITS contains much technical data as well as talk. All in all, combinations of weighting and ITS-specific preprocessing were responsible for the best obtainable results. As discussed in RQ_3, not considering the code and emphasizing the title worked best for each project.

In addition, we compared the values of the fully preprocessed datasets from Table 5 to the same baseline as in Table 7 (only standard preprocessing). This comparison revealed that the preprocessed dataset performs better for different trace and issue types

[13] We also allowed the annotation of the following trace types: I_1 precedes, is parent of, blocks, clones I_2

Table 5. Best Results for Different Issue Types

		$trace_t : I_{feature} \times I_{feature}$			$trace_t : I_{feature} \times I_{bug}$			$trace_t : I_{bug} \times I_{bug}$		
		Results	Alg.	Weights	Results	Alg.	Weights	Results	Alg.	Weights
c:geo	F_1	0.4	BM25	2,1,0,0	0.46	VSM	8,4,2,1	**0.64**	VSM	1,1,1,0
	F_2	0.53	VSM	4,2,1,0	0.41	VSM	8,4,2,1	**0.67**	VSM	1,1,1,0
	$R\,(P)$	1 (0.6)	BM25	1,1,0,0	1 (0.03)	BM25	1,1,0,0	1 (0.04)	VSM	1,1,0,0
Lightt.	F_1	**0.67**	VSM	1,1,0,0	**0.67**	VSM	1,1,1,0	0.33	LSA	8,4,2,1
	F_2	0.56	VSM	1,1,0,0	**0.71**	VSM	1,1,1,0	0.43	VSM	8,4,2,1
	$R\,(P)$	1 (0.02)	BM25	1,1,0,0	1 (0.8)	BM25	1,1,0,0	1 (0.01)	BM25	4,2,1,0
Redm.	F_1	**0.49**	VSM	2,1,0,0	0.29	VSM	4,2,1,0	0.29	VSM	4,2,1,0
	F_2	**0.55**	VSM	2,1,0,0	0.30	VSM	1,1,0,0	0.38	VSM	4,2,1,0
	$R\,(P)$	1 (0.07)	BM25	1,1,0,0	1 (0.03)	BM25	1,1,1,0	0.04 (1)	VSM	1,1,1,0

Table 6. Best Results for Different Trace Types

		$trace_t : I \times I$			$trace_t^{duplicate} : I \times I$		
		Results	Alg.	Weights	Results	Alg.	Weights
c:geo	F_1	0.58	VSM	2,1,0,0	**0.67**	LSA	1,1,0,0
	F_2	0.55	VSM	2,1,0,0	**0.56**	LSA	1,1,0,0
	$R\,(P)$	0.1 (0.03)	BM25+	1,1,1,1	1 (0.11)	BM25	1,1,0,0
Lightt.	F_1	**0.4**	VSM	4,2,1,0	0.18	LSA	1,1,0,0
	F_2	**0.46**	VSM	4,2,1,0	0.36	VSM	2,1,0,0
	$R\,(P)$	0.97 (0.04)	BM25	1,1,1,1	0.97 (0.3)	BM25	1,1,0,0
Redm.	F_1	**0.31**	VSM	1,1,0,0	**0.31**	LSA	1,2,1,1
	F_2	**0.38**	VSM	2,1,0,0	0.36	LSA	1,2,1,1
	$R\,(P)$	0.99 (0.03)	VSM	1,1,1,1	1 (0.01)	LSA	1,1,0,0

Table 7. Best Results per Project (Trace and Issue Type not Distinguished)

		Best Results $trace_t : I \times I$					Baseline
		Results	Alg.	Weights	Std. Pre.	ITS-specific Pre.	Std. Pre. only no weighting
c:geo	F_1	0.58	VSM	2,1,0,0	true	true	0.46 LSA
	F_2	0.55	VSM	2,1,0,0	true	true	0.44 LSA
	$R\,(P)$	1 (0.03)	BM25+	1,1,1,1	false	true	0.99 (0.03) BM25+
Lightt.	F_1	0.4	VSM	4,2,1,0	true	true	0.32 VSM
	F_2	0.46	VSM	4,2,1,0	true	true	0.41 VSM
	$R\,(P)$	0.97 (0.04)	BM25	1,1,1,1	false	false	0.94 (0.03) VSM
Radiant	F_1	0.27	VSM	2,1,0,0	true	true	0.17 LSA
	F_2	0.35	VSM	2,1,0,0	true	true	0.24 VSM
	$R\,(P)$	1 (0.02)	BM25	2,1,0,0	false	false	1 (0.02) BM25
Redm.	F_1	0.31	VSM	2,1,0,0	true	true	0.25 VSM
	F_2	0.38	VSM	2,1,0,0	true	true	0.33 VSM
	$R\,(P)$	0.99 (0.3)	VSM	1,1,1,1	stopword only	false	0.99 (0.03) VSM

as well. We noticed improvements in every case. Most significantly, improvements in both, F_1 and F_2, of over 36% are achieved for $trace_t : I_{bug} \times I_{bug}$ in c:geo and over 10% for $trace_t : I_{feature} \times I_{bug}$ in c:geo. On average, F_1 increased by 19.5% and F_2 by 13.33% for all trace projects and trace types.

Since no BM25 variants performed best, we calculated the improvements in comparison to the baseline from Figure 2. BM25 still performs worse than VSM and LSI. However, the F_2 scores for BM25[+,L] improved by 23% for c:geo, 3% for Lighttpd, 3% for Radiant, and 6% for Redmine.

Overall, the results show that there is neither the best algorithm, nor the best preprocessing for all projects. However, removing code snippets and stack traces (see the term weights for `n-n-n-0` in Table 7) can be considered a good advice. It generally improves the results, especially precision, and has a negative impact of $< 4\%$ on the F_2 measure for Lighttpd in our experiments, only. Also, up-weighting title and down-weighting comments has an overall positive impact. Noticeably, the best measures in Table 7 are computed with the "simplest" algorithm: VSM. Since VSM considers every term of the text that was not removed by preprocessing, we hypothesize that this property is an important factor on ITS data.

7 Threats to Validity

Each GSTM was created by one person only. We tried to minimize this threat by (a) creating and discussing guidelines on how the gold standard should be made and when issues should be seen as related, and (b) peer reviewing the created gold standards by random samples. Although the authors knew the projects or took time to become acquainted with the projects, some traces were hard to decide on. In case of doubt, no trace was inserted in the GSTM. Even though we created rather large GSTMs of 100×100 traces, the GSTMs comprise only small parts of the projects ITSs. Therefore, a generalization from these results cannot be made, although we included about a third of the issues of the Radiant project which is a rather large sample. It gives, however, an indication of the importance of preprocessing and term weighting and shows that ITS data cannot be handled in the same way as structured RAs. In addition to the facts discussed in 6.3, due to the low number of duplicates in our datasets (see Table 2) the low results for duplicates might have occurred by chance. It is important to note that the definitions of *related and* duplicate issues have a major influence on the results. Different definitions would certainly lead to different results since trace matrices are always use-case-dependent.

Finally, OpenTrace creates queries in Apache Lucene to calculate $similarity : I \times I$. This involves data transformations from and to the GATE and OT frameworks. We inspected and enhanced the code very carefully to minimize implementation problems and publish the source code and all data along with this paper.

8 Conclusion and Future Work

In this paper, we presented an evaluation of five IR algorithms for the problem of automated trace retrieval on ITS data. To properly perform this evaluation, four gold standards for 100×100 issues were created. The evaluation considered four open source projects with distinct properties in terms of project size, audience, and so forth. Since the nature of feature descriptions in ITSs is not comparable to requirement artifacts, our results show that algorithms that perform quite well with RAs perform significantly weaker with ITS data. A combination of ITS-specific preprocessing as well as ITS data field-specific term weighting can positively influence the results.

To further improve trace retrieval in ITS, specific NL content needs to be better understood. Our experiment shows that standard IR preprocessing as well as ITS-specific efforts do generally have a positive impact on the results. However, results vary due to the entirely different nature of NL data in different projects. Our extended version of the OpenTrace framework can be used to find good preprocessing and weighting schemes automatically, if a gold standard is available, and it can be extended with other efforts from related work.

References

1. Angius, E., Witte, R.: OpenTrace: An Open Source Workbench for Automatic Software Traceability Link Recovery. 2012 19th Working Conference on Reverse Engineering pp. 507–508 (2012)
2. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval: The Concepts and Technology behind Search. Addison-Wesley Professional, 2 edn. (2011)
3. Borg, M., Runeson, P., Ardö, A.: Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability. Empirical Software Engineering 19(6), 1565–1616 (2014)
4. Chen, X., Hosking, J., Grundy, J.: A Combination Approach for Enhancing Automated Traceability. In: Proceedings of the 33rd Intl. Conference on Software Engineering. pp. 912–915. ACM, Waikiki, Honolulu, HI, USA (2011)
5. Cleland-Huang, J., Settimi, R., Romanova, E., Berenbach, B., Clark, S.: Best Practices for Automated Traceability. Computer 40(6), 27–35 (2007)
6. Cunningham, H., Maynard, D., Bontcheva, K.: Text Processing with GATE (Version 6). University of Sheffield Department of Computer Science (2011)
7. Furnas, G.W., Deerwester, S., Dumais, S.T., Landauer, T.K., Harshman, R.a., Streeter, L.a., Lochbaum, K.E.: Information retrieval using a singular value decomposition model of latent semantic structure. In: Proceedings of the 11th Intl. ACM SIGIR Conference on R&D in Information Retrieval - SIGIR '88. pp. 465–480. ACM Press, New York (1988)
8. Gervasi, V., Zowghi, D.: Mining Requirements Links. In: Proceedings of the 17th Intl. Working Conference on Requirements Engineering: Foundation for Software Quality. vol. 6606 LNCS, pp. 196–201. Springer Berlin / Heidelberg (2011)
9. Gervasi, V., Zowghi, D.: Supporting Traceability Through Affinity Mining. In: IEEE 22nd Intl. Requirements Engineering Conference. pp. 143–152. IEEE (2014)
10. Gotel, O., Cleland-Huang, J., Hayes, J.H., Zisman, A., Egyed, A., Grunbacher, P., Antoniol, G.: The quest for Ubiquity: A roadmap for software and systems traceability research. In: 20th IEEE Intl. Requirements Engineering Conference. pp. 71–80. IEEE (2012)
11. Guo, J., Cleland-Huang, J., Berenbach, B.: Foundations for an expert system in domain-specific traceability. In: 21st IEEE Intl. Requirements Engineering Conference (RE). pp. 42–51. No. 978, IEEE (2013)
12. Heck, P., Zaidman, A.: Horizontal Traceability for Just-In-Time Requirements: The Case for Open Source Feature Requests. Journal of Software: Evolution and Process 26(12), 1280–1296 (2014)

13. Huffman Hayes, J., Dekhtyar, A., Sundaram, S.K.: Advancing Candidate Link Generation for Requirements Tracing: The Study of Methods. IEEE Trans. on SE 32(1) (2006)
14. ISO/IEC/IEEE: Intl. STANDARD ISO/IEC/IEEE 29148:2011 (2011)
15. Lv, Y.: Lower-Bounding Term Frequency Normalization. ACM Conference on Information and Knowledge Management pp. 7–16 (2011)
16. Lv, Y., Zhai, C.: When documents are very long, BM25 fails! In: Proceedings of the 34th Intl. ACM SIGIR Conference on R&D in Information Retrieval - SIGIR '11. p. 1103. No. I, ACM Press, New York (2011)
17. Manning, C.D., Raghavan, P., Schütze, H.: Introduction to Information Retrieval. Cambridge University Press, 1 edn. (2008)
18. Merten, T., Mager, B., Bürsner, S., Paech, B.: Classifying unstructured data into natural language text and technical information. In: Proceedings of the 11th Working Conference on Mining Software Repositories - MSR 2014. pp. 300–303. ACM Press, New York (2014)
19. Merten, T., Mager, B., Hübner, P., Quirchmayr, T., Bürsner, S., Paech, B.: Requirements Communication in Issue Tracking Systems in Four Open-Source Projects. In: 6th Intl. Workshop on Requirements Prioritization and Communication (RePriCo). pp. 114–125. CEUR Workshop Proceedings (2015)
20. Natt och Dag, J., Gervasi, V.: Managing Large Repositories of Natural Language Requirements. In: Engineering and Managing Software Requirements, pp. 219–244. Springer-Verlag, Berlin/Heidelberg (2005)
21. Nguyen, A.T., Nguyen, T.T., Nguyen, H.A., Nguyen, T.N.: Multi-layered approach for recovering links between bug reports and fixes. Proceedings of the ACM SIGSOFT 20th Intl. Symposium on the Foundations of Software Engineering - FSE '12 p. 1 (2012)
22. Niu, N., Mahmoud, A.: Enhancing candidate link generation for requirements tracing: The cluster hypothesis revisited. In: 20th IEEE Intl. Requirements Engineering Conference. pp. 81–90. IEEE (2012)
23. Oliveto, R., Gethers, M., Poshyvanyk, D., De Lucia, A.: On the Equivalence of Information Retrieval Methods for Automated Traceability Link Recovery. In: 2010 IEEE 18th International Conference on Program Comprehension. pp. 68–71. IEEE (jun 2010)
24. Paech, B., Hubner, P., Merten, T.: What are the Features of this Software? In: ICSEA 2014, The Ninth International Conference on Software Engineering Advances. pp. 97–106. IARIA XPS Press (2014)
25. Robertson, S.E., Walker, S., Hancock-Beaulieu, M., Gull, A., Lau, M.: Okapi at TREC. In: Proceedings of The First Text REtrieval Conference, TREC 1992. vol. Special Pu, pp. 21–30. National Institute of Standards and Technology (NIST) (1992)
26. Runeson, P., Alexandersson, M., Nyholm, O.: Detection of duplicate defect reports using natural language processing. In: Intl. Conference on SE. pp. 499–508 (2007)
27. Salton, G., Wong, A., Yang, C.S.: A vector space model for automatic indexing. Communications of the ACM 18(11), 613–620 (1975)
28. Skerrett, I., The Eclipse Foundation: The Eclipse Community Survey 2011 (2011)
29. Sultanov, H., Hayes, J.H.: Application of reinforcement learning to requirements engineering: requirements tracing. In: 21st IEEE Intl. Requirements Engineering Conference. pp. 52–61. IEEE (2013)
30. Wang, X.W.X., Zhang, L.Z.L., Xie, T.X.T., Anvik, J., Sun, J.S.J.: An approach to detecting duplicate bug reports using natural language and execution information. 2008 ACM/IEEE 30th Intl. Conference on Software Engineering pp. 461–470 (2008)