

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

Thorsten Merten¹(✉), Daniel Krämer¹, Bastian Mager¹, Paul Schell¹,
Simone Bürsner¹, and Barbara Paech²

¹ Department of Computer Science, Bonn-Rhein-Sieg University of Applied Sciences,
Sankt Augustin, Germany

{thorsten.merten,simone.buersner}@h-brs.de,
{daniel.kraemer.2009w,bastian.mager.2010w,
paul.schell.2009w}@informatik.h-brs.de

² Institute of Computer Science, University of Heidelberg, 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

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:

$$R = \frac{\text{CorrectLinks} \cap \text{RetrievedLinks}}{\text{CorrectLinks}}, \quad P = \frac{\text{CorrectLinks} \cap \text{RetrievedLinks}}{\text{RetrievedLinks}} \quad (2)$$

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: 

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

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 1³ 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.

³ 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.

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 assurance⁵ 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 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.

⁵ Dag and Gervasi [20] surveyed automated approaches to improve the NL quality.

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)	~ 3850	~ 2900	~ 320	~ 19.000
Open issues	~ 450	~ 500	~ 50	~ 4500
Closed issues	~ 3400	~ 2400	~ 270	~ 14.500
Sample size	100 ≈ 3%	100 ≈ 3%	100 ≈ 30%	100 < 1%
Sampled issues with link	~ 50%	~ 20%	~ 12%	~ 70%
Issues labeled explicitly as Feature or Bug in sample	25F/26B	30F/70B	0F/0B	31F/61B
Project size (in LOC)	~ 130,000	~ 41,000	~ 33,000	~ 150,000

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). *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].

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 Sect. 2.3) or when an issue should be treated as duplicate (the description of both issues describes exactly the same bug or requirement).

and
on to
next
page

Table 2. Extracted traces vs. gold standard

# of relations	Projects			
	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
$0.6 \leq r < 0.7$	$0.7 \leq r < 0.8$	$r \geq 0.8$
$0.2 \leq p < 0.3$	$0.3 \leq p < 0.4$	$p \geq 0.4$
$0.2 \leq F_1 < 0.42$	$0.42 \leq F_1 < 0.53$	$F_1 \geq 0.53$
$0.43 \leq F_2 < 0.55$	$0.55 \leq F_2 < 0.66$	$F_2 \geq 0.66$

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.

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 Lucene⁷ 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

⁷ <https://lucene.apache.org>.

Table 4. Data fields weights (l), algorithms and preprocessing settings (r)

Weight				Rationale / Hypothesis	Algorithm	Settings
Title	Description	Comments	Code			
1	1	1	1	Unaltered algorithm	BM25	Pure, +, L
1	1	1	0	– without considering code	VSM	TF-IDF
1	1	0	0	– also without comments	LSI	<i>cos</i> measure
2	1	1	1	Title more important	Preprocessing	
2	1	1	0	– without considering code	Settings	
1	2	1	1	Description more important	<i>Standard</i>	
1	1	1	2	Code more important	Stemming	on/off
8	4	2	1	Most important information first	Stop Word Removal	on/off
4	2	1	0	– without considering code	<i>ITS-specific</i>	
2	1	0	0	– also without comments	Noise Removal	on/off
					Code Extraction	on/off

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 download⁸.

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 Sect. 2) was removed and the regions marked as 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).

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 Sect. 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

⁸ <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.



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.

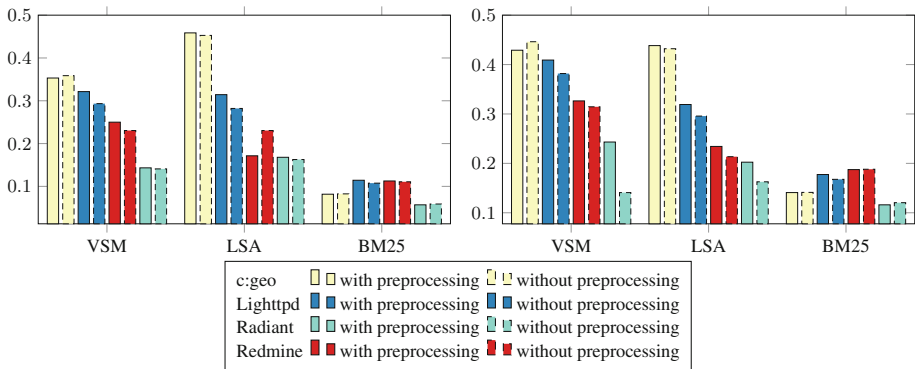


Fig. 2. Best F_1 (left) and F_2 (right) scores for every algorithm

Although results decrease slightly in a few cases, the negative impact is negligible. Therefore, the remaining measurements are reported with the standard preprocessing techniques enabled⁹.

⁹ In addition, removing stop words and stemming is considered IR best practices, e.g. [2, 17].