On the Reduction of Verbose Queries in Text Retrieval Based Software Maintenance

Oscar Chaparro, Andrian Marcus
The University of Texas at Dallas, Richardson, TX, USA
ojchaparroa@utdallas.edu, amarcus@utdallas.edu

ABSTRACT

We argue that verbose queries used for software retrieval contain many terms that follow specific discourse rules, yet hinder retrieval. We report the results of an empirical study on the effect of removing such terms from verbose queries in the context of Text Retrieval-based concept location. In the study, we remove terms from 424 queries, generated from bug reports of nine open source systems. Removing the terms leads to substantial improvement in retrieval: 73% of the queries are improved, leading to 21.8% and 13.4% gain in terms of MRR and MAP, respectively. Such improvement is larger than that of many more sophisticated state-of-the-art approaches. The results show promise and the future challenge lies with automatically identifying the terms to be removed from the verbose queries.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*Restructuring, reverse engineering and engineering*

Keywords

Query Reduction, Software Maintenance, Text Retrieval

1. INTRODUCTION

Researchers developed Text Retrieval (TR)-based techniques to support more than 30 software engineering tasks [2], such as, traceability link recovery or concept location in software. Automatic TR-based approaches usually use as input the complete text of software artifacts as queries. For example, many automatic TR-based concept location approaches, use the title and complete description of bug reports as queries [15, 17, 20, 22, 23], whereas, traceability link recovery techniques often use complete requirements or use cases [4, 7, 8]. Such artifacts have a well defined format [3, 5] and audiences, which impose specific discourse rules [13]. In other words, these artifacts are not meant to be used as queries for software artifact retrieval. For example, bug reports usually describe the observed and expected software behaviors, steps to reproduce the bug, code examples, *etc.* [3]. Often times the bug descriptions use generic phrases, such as, "the application crashes" or "the web page is not loaded in the browser". Such terms are likely to negatively impact the performance of code retrieval.

The traditional way of handling user queries, including verbose ones, in software engineering applications, is through document/query preprocessing and query reformulation. Preprocessing is a common step in the retrieval process that includes: stop word removal, common English and programming language terms removal, code identifiers splitting, stemming, spellchecking, *etc.* [7, 15]. Often, the queries remain verbose even after preprocessing. Query reformulation approaches have mainly focused on adding terms to queries (*a.k.a.* query expansion) [9, 11], such as, synonyms, and selecting/boosting key terms [6], rather than on removing noisy words (*a.k.a.* query reduction). In this paper, we show how the reduction of verbose queries has the potential to substantially improve the performance of TR-based concept location.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

ICSE '16 May 14-22, 2016, Austin, TX, USA

© 2016 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-4205-6/16/05.

DOI: http://dx.doi.org/10.1145/2889160.2892647

2. EMPIRICAL STUDY

The *goal* of our empirical study is to compare the retrieval accuracy achieved by the reduced queries with the one obtained with the original queries. Our *purpose* is to determine the effect of removing terms from queries on the performance of a traditional TR-based concept location technique, *i.e.*, using Lucene [12], to help developers locate bugs in source code. In consequence, we formulate the following research question:

Do reduced queries improve de accuracy of TR-based concept location compared to queries with no modification?

2.1 Context and Planning

The *context* of the study is represented by 424 bug reports marked as fixed, from nine open source systems, used in recent work [17] (see Table 1). The bug reports are used as queries to retrieve classes that need to change. Each query is created by concatenating the title and description of a bug report. Code documents (*i.e.*, classes) are created from identifiers, comments and literals. Queries and code documents are normalized using identifier splitting, special characters removal, common English stop words and programming keywords removal, and stemming [19]. We also remove code snippets, identifier references, and execution traces from the queries (using an Island Parser [18]), as this information

Table 1: **Systems used in the empirical study**

System	# of bug reports	# of classes	# of terms per query[a]
BookKeeper 4.1.0	40	587	14.0 (11.5)
Derby 10.9.1.0	96	3,139	21.1 (17)
Lucene 4.0	34	5,901	15.2 (12)
Mahout 0.8	30	3,260	21.0 (17.5)
OpenJPA 2.2.0	18	4,994	25.3 (23)
Pig 0.11.1	48	2,510	17.2 (14)
Solr 4.4.0	55	6,486	20.0 (16)
Tika 1.3	23	582	20.0 (17)
ZooKeeper 3.4.5	80	697	21.0 (16)
Total	*424*	*28,156*	*19.4 (16)*

[a]. Average values, and in parenthesis, median values

Table 2: Maximum retrieval performance achieved by reduced queries in comparison with the original queries

Queries	Avg Effect.[a]	MRR	MAP	HIT@1
Original	159.5 (8)	31.7%	21.9%	21.7%
Reduced	103 (2)	53.5%	35.3%	41.7%

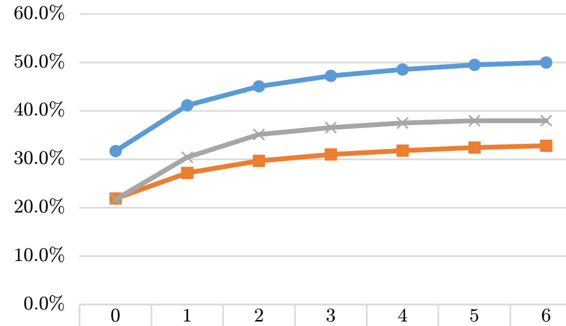
[a]. In parenthesis, median *Effectiveness* values

is likely to contain explicit references to the code, thus reducing bias [14, 21]. After preprocessing, the queries still remain verbose, as the average query length is 19.4 terms (see Table 1).

To evaluate the performance of the TR-based approach (*i.e.*, Lucene), we compute a set of metrics against a gold set, which contains the relevant code documents for each query. Existing classes of the software systems that were modified to fix each bug represent the relevant code documents. We manually filtered out those classes with changes that were not intended to fix the bug described in the reports. We utilize standard metrics previously used in concept location research [10, 16]. We use *Effectiveness*, *i.e.*, the best rank obtained for a query; *Mean Reciprocal Rank* (MRR), a statistic that measures the aggregate quality of the ranking of a retrieval approach and is computed as the average between the reciprocal *Effectiveness* of a set of queries; *Mean Average Precision* (MAP), another aggregate measure that reflects how well all the changed documents rank; and HIT@1, the number of queries with one relevant document retrieved in the top of the ranked list (see [17, 22] for the metric definitions).

In order to identify the query terms to be removed we perform the following procedure. For a particular query q, we obtain its baseline *Effectiveness* by running q with Lucene without any modification. Then, for each term t in q, we create a new query q_t by removing t from q. We run q_t and measure the *Effectiveness* achieved by the query. Finally, we mark a term t as *to-remove* if the reduced query q_t achieves a better (*i.e.*, lower) *Effectiveness* than the baseline *Effectiveness*. To obtain the reduced queries, we sort the marked terms in descending order, by the magnitude of improvement in *Effectiveness*, and remove them one by one from the original query, starting from the top one. We measure and report the performance of each reduced query (see [1]). This procedure is repeated for every query in our dataset.

Figure 1: Retrieval performance when k terms are removed from the queries



	0	1	2	3	4	5	6
MRR	31.7%	41.2%	45.1%	47.2%	48.6%	49.5%	50.0%
MAP	21.9%	27.2%	29.7%	31.0%	31.8%	32.4%	32.8%
HIT@1	21.7%	30.4%	35.1%	36.6%	37.5%	38.0%	38.0%

\# of terms removed

2.2 Results

Table 2 summarizes the results obtained for the original and reduced queries, when all the marked terms are removed. 26% of the queries were not reduced, as no term was marked for reduction. As Table 2 reveals, most of these (21.7% - see HITS@1) already have *Effectiveness* one (1), so removing any term cannot lead to improvement. As for the improvements, nearly 73% of the queries are improved via the reduction, reaching up to 21.8%, 13.4% and 20% overall gain in terms of MRR, MAP and HIT@1, respectively. We note that the improvement is higher or comparable with the results reported by state-of-the-art research, where multiple sources of information are used [15, 22, 24]. The reduced queries also achieve a two (2) median effectiveness, *i.e.*, for 50% of the queries, the first relevant document is retrieved in the first two positions. For the 74% of the queries where at least one term is removed, 6.6 out of 18.2 query terms are removed, in average (*i.e.*, 36.3% of the queries' length). It is interesting to note that 1% of the queries had terms marked for removal, yet when all of them were removed, the results did not improve. Such cases need further investigation.

We also report the performance trend when the terms are removed one by one from the original queries (*i.e.*, when 0 terms are deleted). Fig. 1 shows that such trend follows a monotonic increasing behavior, having higher value changes when few terms are removed. The curve then slowly grows as the number of terms removed increases. The improvement is substantial when one term is deleted (9.5% MRR, 5.3% MAP and 8.7 HIT@1).

3. CONCLUSION AND FUTURE WORK

Our empirical study indicates that verbose queries used in TR-based software maintenance can be substantially improved via reduction. The main challenge for future work is automatically identifying the terms that should be removed to achieve retrieval improvement. Machine learning techniques could be used to identify such terms, based on statistical and semantic features of the terms.

Acknowledgments

This research was supported in part by the following NSF grants: CCF-1526118 and CCF-1514460.

4. REFERENCES

[1] Online replication package: https://seers.utdallas.edu/projects/query-reduction-poster.

[2] V. Arnaoudova, S. Haiduc, A. Marcus, and G. Antoniol. The use of text retrieval and natural language processing in software engineering. In *Proceedings of the 37th International Conference on Software Engineering*, pages 949–950, 2015.

[3] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann. What makes a good bug report? In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE'16)*, pages 308–318, 2008.

[4] M. Borg, P. Runeson, and A. Ardö. Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability. *Empirical Software Engineering*, 19(6):1565–1616, 2014.

[5] S. Breu, R. Premraj, J. Sillito, and T. Zimmermann. Information needs in bug reports: Improving cooperation between developers and users. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'10)*, pages 301–310, 2010.

[6] G. Capobianco, A. D. Lucia, R. Oliveto, A. Panichella, and S. Panichella. Improving ir-based traceability recovery via noun-based indexing of software artifacts. *Journal of Software: Evolution and Process*, 25(7):743–762, 2013.

[7] A. De Lucia, A. Marcus, R. Oliveto, and D. Poshyvanyk. Information retrieval methods for automated traceability recovery. In J. Cleland-Huang, O. Gotel, and A. Zisman, editors, *Software and Systems Traceability*, pages 71–98. Springer, 2012.

[8] D. Diaz, G. Bavota, A. Marcus, R. Oliveto, S. Takahashi, and A. De Lucia. Using code ownership to improve ir-based traceability link recovery. In *Proceedings of the IEEE 21st International Conference on Program Comprehension (ICPC'13)*, pages 123–132, 2013.

[9] T. Dietrich, J. Cleland-Huang, and Y. Shin. Learning effective query transformations for enhanced requirements trace retrieval. In *Proceedings of the IEEE/ACM 28th International Conference on Automated Software Engineering (ASE'13)*, pages 586–591, 2013.

[10] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk. Feature location in source code: A taxonomy and survey. *Journal of Software: Evolution and Process*, 25(1):53–95, 2012.

[11] S. Haiduc, G. Bavota, A. Marcus, R. Oliveto, A. De Lucia, and T. Menzies. Automatic query reformulations for text retrieval in software engineering. In *Proceedings of the 35th IEEE/ACM International Conference on Software Engineering (ICSE'13)*, pages 842–851, 2013.

[12] E. Hatcher and O. Gospodnetic. *Lucene in Action*. Manning Publications, 2004.

[13] A. J. Ko, B. A. Myers, and D. H. Chau. A linguistic analysis of how people describe software problems in bug reports. In *Proceedings of the IEEE Conference on Visual Languages and Human-Centric Computing (VL/HCC'06)*, pages 127–134, 2006.

[14] P. S. Kochhar, Y. Tian, and D. Lo. Potential biases in bug localization: Do they matter? In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering (ASE'14)*, pages 803–814, 2014.

[15] T.-D. B. Le, R. J. Oentaryo, and D. Lo. Information retrieval and spectrum based bug localization: Better together. In *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE'15)*, pages 579–590, 2015.

[16] A. Marcus and G. Antoniol. On the use of text retrieval techniques in software engineering. In *Proceedings of 34th IEEE/ACM International Conference on Software Engineering (ICSE'12)*, 2012.

[17] L. Moreno, J. Treadway, A. Marcus, and W. Shen. On the use of stack traces to improve text retrieval-based bug localization. In *Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME'14)*, pages 151–160, 2014.

[18] L. Ponzanelli, A. Mocci, and M. Lanza. Stormed: Stack overflow ready made data. In *Proceedings of the 12th Working Conference on Mining Software Repositories (MSR'15)*, page to appear, 2015.

[19] M. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[20] R. Saha, M. Lease, S. Khurshid, and D. Perry. Improving bug localization using structured information retrieval. In *Proceedings of the 28th International Conference on Automated Software Engineering (ASE'13)*, pages 345–355, 2013.

[21] Q. Wang, C. Parnin, and A. Orso. Evaluating the usefulness of ir-based fault localization techniques. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis (ISSTA'15)*, pages 1–11, 2015.

[22] S. Wang and D. Lo. Version history, similar report, and structure: Putting them together for improved bug localization. In *Proceedings of the 22nd International Conference on Program Comprehension (ICPC'14)*, pages 53–63, 2014.

[23] S. Wang, D. Lo, and J. Lawall. Compositional vector space models for improved bug localization. In *Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME'14)*, pages 171–180, 2014.

[24] X. Ye, R. Bunescu, and C. Liu. Mapping bug reports to relevant files: A ranking model, a fine-grained benchmark, and feature evaluation. *IEEE Transactions on Software Engineering*, PP(99):1–1, 2015.