Structured Source Retrieval for Improving Software Search during Program Comprehension Tasks

Brian Eddy
Department of Computer Science
The University of Alabama
bpeddy@crimson.ua.edu

Abstract

During the software maintenance and evolution phase, the majority of a developer's time is spent on programming comprehension tasks. Feature location (i.e., finding the first location to make a modification), impact analysis (i.e., determining what and to what extent a program is affected by a change), and traceability (i.e., determining where requirements are implemented in the program), are all examples of such tasks. Recent research in the area of program comprehension has focused on using textual information, structural information (i.e., information regarding the creation and use of objects and methods within the code), and execution traces to develop tools that ease the burden on developers and decrease the time spent in each task. Furthermore, new studies in automating these tasks have started using text retrieval techniques, such as the vector space model (VSM), latent semantic indexing (LSI), and latent Dirichlet allocation (LDA) for searching software. This doctoral symposium summary presents two promising areas for improving existing techniques by combining structural information with text retrieval. The first is a methodology for evaluating the usefulness of text obtained from a program by looking at the structural location of terms (e.g., method name, comments, identifiers). The second focuses on improving the existing text retrieval approaches by providing more flexible queries (i.e., search strings). These two areas are complementary to each other and may be combined.

1. Motivation for Structured Source Retrieval

Understanding a software system's implementation is a crucial part of a developer's job. When developers are tasked with changing the source code of a large or unfamiliar system, they must spend considerable time and effort on program comprehension activities to gain the knowledge needed to implement, correct, and complete changes. Before any changes can be made, bugs fixed, or features added, the developer must first understand the system's implementation and locate source code elements specific to the current task.

Software maintenance and evolution accounts for 60-80% [1, 5] of the cost and effort during the software life cycle, and during this time, over half [8] of developer effort is taken up in trying to understand the software system. One common task involves locating a source code element that is required for a change or modification. This process, known as feature location, becomes impractical to perform manually as the scale of modern software systems increases. Existing search tools such as keyword searches and regular expressions are limited as such techniques do not allow for searching of synonyms (missing relevant results) or searching for only a single meaning of a term when a term has multiple meanings (resulting in irrelevant results) [4]. Given these problems and the increasing size and complexity of software systems, the need for more advanced tools to aid in program comprehension is evident.

Thus, techniques and tools that can reduce the effort required for these tasks are key to minimizing software costs. Many recent studies into semi-automated techniques for program comprehension tasks are based on text retrieval (TR) methods that focus on producing corpora of text extracted from source code and performing searches over the newly created model. Common TR-based tools in the literature incorporate models such as latent semantic indexing (LSI) [4] and latent Dirichlet allocation (LDA) [3].

Previous research into TR-based techniques have attempted to combine textual information with multiple sources of structural (e.g., dependency graphs) and dynamic information (e.g., execution traces). By combining textual information with structural information and dynamic information, researchers have found an improvement over using textual information alone for some systems [10–13]. There are two existing problems with this research area. First, the research has focused on latent semantic indexing (LSI) and the vector space model (VSM), discussed in more detail later. However, more sophisticated techniques such as latent Dirichlet allocation (LDA) and other topic-modeling techniques (e.g., probabilistic LSI, Pachinko allocation, associative clustering) have been shown to produce useful results in other domains along with program comprehension tasks. Second, there has not been much investigation into the importance of each structural component (e.g., whether method calls are as important as method names).

The goal of the research presented in this doctoral symposium summary is to investigate how new structural weighting schemes (i.e., schemes that give varying weights to different structural components) affects the results of text retrieval as it applies to software maintenance tasks. The research plan will study two different techniques. The first combines structural weighting with LDA as an ad-hoc preprocessing step. The second integrates structural weighting directly into a structured text retrieval approach based on language modeling. We believe this work will lead to more relevant results and as discussed in Section 3.2, lead to a more flexible retrieval process for the developer.

SPLASH '14, Oct 20-24 2014, Portland, OR, USA.
Copyright © 2014 ACM 978-1-4503-3208-8/14/10...$15.00.
http://dx.doi.org/10.1145/2660252.2660253

2. Problems of Interest

We are currently working on studying the effects of structural weighting schemes on LDA and finishing the development of tool support for the structured source retrieval approach. The main problems are discussed in this section.

2.1 RP1: Determining the effects of structural weighting schemes on LDA.

TR models operate on corpora. A corpus is a set of documents containing all text associated with the document after it has undergone a set of text transformations (i.e., preprocessing steps). Each document in one of these corpora contains the text associated with a single source code entity, typically a method. TR techniques such as LSI and VSM have shown to have their results improved when combined with additional structural and dynamic information [7, 9, 10, 13]. We propose to investigate the effect of different weighting schemes for LDA. Term weighting is a common preprocessing step. These weighting schemes will take into account the structural location (e.g., parameter, comment, method name) of a term in source documents.

One criticism of the use of advanced topic modeling approaches on source code is that terms are more sparse than in natural language documents. For instance, the most relevant topic to a method may be the one that describes the method's behavior. However, terms for that topic may be limited to the method name and parameters and it is common for the method name to be limited to the method's signature. In such a case, placing higher importance on the terms in the method name may result in higher probability of a document being associated with the correct topic(s). However, this does not necessarily indicate that other terms should be disregarded. Emphasizing certain terms (e.g., method names) while deemphasizing others (e.g., method calls) may lead to a better topic model.

2.2 RP2: Determining how various weighting schemes affect structured source retrieval in language models for software maintenance tasks.

RP1 focuses on a traditional TR technique (LDA). Traditional techniques treat documents as unordered collections of terms without structure. However, not all documents are unstructured. For instance, a scientific article may be broken into the title, the abstract, the sections, the paragraphs, and the sentences. Words may appear in multiple components of the document or in a single component of the document. The approach of performing information retrieval by breaking documents into fragments based on the structure of the document, and either returning the most relevant fragments as a result of a query or using the fragments to find the most relevant documents, is known as structured document retrieval [6]. The structure of a document may be either explicitly defined using a mark-up language (e.g., XML) or derived.

We have investigated an approach to structured document retrieval on source code by deriving structure from the position and origin of the terms (e.g. method signature, method body, comments). Structured source retrieval techniques allow for new querying methodologies. For instance, a developer may have an understanding of what terms relate to method names and class names, and what terms refer to variables or fields. A developer might also have expectations of what context a term is used. Allowing a more robust query system that allows developer input may increase the likelihood of returning relevant results. Current queries used in software search do not allow for order of terms or for varying levels of emphasis on different structural components.

We will first finish a study of the benefits and consequences of the structured source retrieval technique and then investigate the combination of structured source retrieval with structural weighting from RP1.

3. The Approach

In this section, we describe the approach to structural weighting as well as our approach to structured document retrieval. We will first study the effects of structural weighting on LDA, then develop the tools needed for the structured document retrieval approach and study the effects of structural weighting on that technique. Once both parts are completed, we will compare the results of both approaches.

3.1 Structural Weighting

Structural weighting was introduced for LDA by Bassett and Kraft [2]. In their study, they focused on various weighting schemes of method names and method calls by changing term counts during the creation of the corpus. They found that by changing the count of certain terms, it was possible to achieve more accurate results with LDA.

There is still work to be completed on this idea. We will expand upon the previous research by focusing on other terms that will likely lead to an overall improvement in relevant results. There are a number of different structural components at both the method and the class level. Prior research has focused mainly on the method level for text retrieval tasks. At the method level, different weights can be placed on the following components: parameters, string literals, local variables, method calls, annotations, class or interface references, method names, in-line comments, block comments. Each of these components may be weighted individually or as a group with other items on the list. Previous experience has shown that there are two things that need to be considered: the individual component alone and the interaction between components.

A document in LDA is a collection of terms appearing in that document. Weights will be introduced to LDA through the use of scalar multiples that increase or decrease the number of times certain terms appear in these collections. A weighting scheme is then expressed as the scalar multiple for each component. By using scalar multiples and weighting the components differently, LDA may be modified to increase the probablity that certain terms will be associated with a particular document. We believe certain elements are more important to a method. For instance, method names are more important than method calls as they describe the behavior of the method itself, while a method call may be to a supporting object such as a logger or only a sub step of the method's behavior. By raising the weights on these terms we emphasize what we believe to be important, while deemphasizing what we believe to be unimportant. This produces results where methods with terms appearing in the method names are places higher than methods with terms appearing in the method calls.

Investigating this idea will start by focusing on a small subset of the components that are believed to have a high likelihood to influence the results. The initial focus will be on the leading comments, method names, parameters, and body comments. It is believed that leading comments are often used to explain the purpose of the method in natural language and therefore have a high relevance to the software search. Parameters are the inputs to the functions. Combined with the method name, they help to clarify the main responsibility of the method or help clarify which method is most relevant to a query amongst a set of overloaded methods. We will study these components by first identifying appropriate levels for the scalars applied to each component, then systematically changing the weights of each of these terms individually and as linear combinations, then performing feature location on multiple open source software systems (jEdit, JabRef, Eclipse). We will use statistical analysis to identify significance of an effective change and to identify interactions among the components.

3.2 Structured Source Retrieval

Structural Weighting offers the possibility of improvement in traditional TR-based techniques. However, structured source retrieval differs from traditional techniques by building structured documents from source code.

Structured document retrieval divides the terms in a document into multiple components where each component is a structural field of the document. For instance, a document could be divided into the title and the body of the document. For source code where documents are typically methods in the software system, the document could be divided in multiple different ways. One method could split the method signature, method body, and comments into different components. Another might only use the method signature and method body. Yet, another might split the method signature, method identifiers, method literals, and method comments. While there are several ways to perform this splitting, a finer granularity leads to more flexibility when querying but also leads to more complexity in the retrieval model. Research will need to be done to determine the best way of forming structured method documents.

Once the source code has been converted into structured documents, tools exist for indexing such documents. Indri[1] is a search engine developed as part of the Lemur project between the University of Massachusetts and Carnegie Mellon University. The search engine supports structured query documents and provides the user of the system with a flexible model for defining fields and other attributes of a document in the corpus. The system uses a combination of language modeling and inference networks as the search engine's retrieval model. We will adapt our source code model to be searchable by Indri's search engine.

The Indri query language allows for a wide variety of options. The simplest queries in Indri take the form:

#combine(side1 computes Point)

The # signifies a query, *combine* means to search for the terms together in a document, while the query is provided in the parentheses. For such a query, each term in the query is given equal weighting and the query is issued across all fields. This query does not make use of the structured document, but instead uses the document as a collection of words similar to the traditional TR approaches. A more advanced query would be of the form:

#weight(2.0 #combine[signature](area) 1.0
#combine[body](area))

In the example query, the developer has given greater weight to documents with "area" appearing in the method signature versus the method body. Perhaps the developer knows the method they want computes an area, so they believe that "area" is likely to be in the method name. They want to see methods with "area" in the method name before other possible choices. In the example given, area has twice the effect on the final score when it appears in the signature compared to the body. If we treat b as the belief score (i.e., how likely we believe the document will return this word) then an example of the overall score for a document given the example query might be $0.67 * log(b(\#combine[signature](area))) + 0.33 * log(b(\#combine[body](area)))$.

By weighting the query, developers are more likely to retrieve the results they want. While developers may manually weight their queries, we also wish to identify queries that may return higher results for developers that are less familiar with the system or suggest queries that may produce more relevant results to the developer. The focus of this research is on using the weighted queries to produce more relevant results.

[1] http://www.lemurproject.org/indri/

4. Evaluation Methodology

To evaluate our structural weighting schemes and our structured source retrieval approach, we will use established benchmarks in the field of feature location. We will begin by repeating the study conducted by Bassett and Kraft [2]. Then we will expand our study to look at terms from the method signature, the leading comments, and body comments. We believe these components are the most likely to lead to more relevant results. We will perform statistical testing to look for significant effects between different weighting schemes and factor analysis to identify any interactions.

Once we have completed our work on LDA, we will finish development of the tools needed for our structured document retrieval technique. We will use a methodology similar to that of studying structural weighting on LDA to study the effects of different weighted queries. We will then compare the results of the two different approaches (structural weighting with LDA and structured source retrieval). Finally, we will assess whether either of these techniques results in an improvement over traditional TR-based techniques on impact analysis.

References

[1] G. Alkhatib. The maintenance problem of application software: an empirical analysis. *Journal of Software Maintenance: Research and Practice*, 4(2):83–104, 1992.

[2] B. Bassett and N. A. Kraft. Structural information based term weighting in text retrieval for feature location. In *Program Comprehension (ICPC), 2013 IEEE 21st International Conference on*, pages 133–141. IEEE, 2013.

[3] D. Blei, A. Ng, and M. Jordan. Latent Dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[4] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41:391–407, 1990.

[5] L. Erlikh. Leveraging legacy system dollars for e-business. *IEEE IT Pro*, pages 17–23, May/June 2000.

[6] M. Lalmas and R. Baeza-Yates. *Structured Document Retrieval*. Springer US, 2009.

[7] D. Liu, A. Marcus, D. Poshyvanyk, and V. Rajlich. Feature location via information retrieval based filtering of a single scenario execution trace. In *Proceedings of the 22nd International Conference on Automated Software Engineering*, pages 234–243, 2007.

[8] H. Müller, J. Jahnke, D. Smith, M.-A. Storey, S. Tilley, and K. Wong. Reverse engineering: A roadmap. In *Proceedings of the Future of Software Engineering*, pages 47–60, June 2000.

[9] D. Poshyvanyk, Y. Gueheneuc, A. Marcus, G. Antoniol, and V. Rajlich. Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval. *IEEE Transactions on Software Engineering*, 33(6):420–432, June 2007.

[10] M. Revelle, B. Dit, and D. Poshyvanyk. Using data fusion and web mining to support feature location in software. In *Proceedings of 18th IEEE International Conference on Program Comprehension*, pages 14–23, Braga, Portugal, July 2010.

[11] G. Scanniello and A. Marcus. Clustering support for static concept location in source code. In *Proceedings of the 19th IEEE International Conference on Program Comprehension*, 2011.

[12] P. Shao and R. K. Smith. Feature location by ir modules and call graph. In *Proceedings of the 47th Annual Southeast Regional Conference*, pages 70:1–70:4, Clemson, South Carolina, 2009. ISBN 978-1-60558-421-8.

[13] W. Zhao, L. Zhang, Y. Liu, J. Sun, and F. Yang. SNIAFL: Towards a static noninteractive approach to feature location. *ACM Transactions of Software Engineering Methodologies*, 15(2):195–226, 2006.