A TraceLab-Based Solution for Identifying Traceability Links using LSI

Nouh Alhindawi, Omar Meqdadi, Brian Bartman, Jonathan Maletic
Department of Computer Science
Kent State University
Kent Ohio 44242
{nalhinda, omeqdadi, bbartman, jmaletic}@kent.edu

***Abstract*—An information retrieval technique, latent semantic indexing (LSI), is used to automatically identify traceability links from system documentation to program source code. The experiment is performed in the TraceLab framework. The solution provides templates and components for building and querying LSI space and datasets (corpora) that can be used as inputs for these components. The proposed solution is evaluated on traceability links already discovered by mining adaptive commits of the open source system KDE/Koffice. The results show that the approach can identify of traceability links with high precision using TraceLab components.**

***Index Terms*—Component, Traceability links, TraceLab, Information Retrieval, Adaptive Changes.**

I. Introduction

Traceability between software artifacts is essential for many large-scale systems. Especially in the case of safety critical system and organizations that adopt standards such as SIL (Safety Integrity Level) or IEC 61508 from the European Functional Safety standards. However it is often difficult to directly support and maintain traceability links over the lifetime of a software system.

Some of this difficult rests in the fact that it is difficult for researchers to develop and share data sets on traceability. It is also difficult to compare experiments on uncovering traceability links. To address this problem an experimental sandbox is being developed to support traceability research. TraceLab [1, 2] is an environment where traceability experiments can be easily constructed and reproduced using reusable components. It uses a visual modeling environment to set up experiments with the components. It also allows experiments to be easily repeated by other researchers. TraceLab was created at DePaul University with collaborating partners at Kent State University, University of Kentucky, and the College of William and Mary.

In the work presented here we demonstrate how TraceLab can be used to run experiments, similar to those found in [8, 9], of using the information retrieval method, Latent Semantic Analysis (LSI) to uncover links between documents and source code. The goal is to show how TraceLab can be used and how the reusable component can be utilized to build experiments.

We also present a means to generate a set of traceability data in a semi-automated manner via frequent itemset mining. We utilize some manually generated information along with frequent itemset mining to uncover a set of traceability links for a specific type of software maintenance task. Here we look at a particular adaptive maintenance task that involves the migration of an API.

The results of the experiment align well with the original findings and form a basis for running a variety of experiments to test hypothesis on different parameters.

The remainder of the paper is organized as follows. Section II describes the components we developed for TraceLab in addition to some existing ones that we utilized. Section III details the traceability data we used as our golden set and how we derived it. Section IV gives the experiment and results. Finally, conclusions and future work are presented.

II. TraceLAB Components

For our experiment here, we created several components in order to use LSI [3], as shown in Fig. *1*. Each of the created components was designed to use the existing TraceLab types, so they can be easily integrated into other experiments as needed. Our components are written in C# and C++ .We created the following components.

A. LSI Space Builder

This component is used to construct the LSI space for a given corpus, as shown in Fig. *2*. For an input, it takes a set of document names and documents, which make up a corpus. It outputs the LSI space up to a specified rank; a dictionary of document titles and a dictionary of vocabulary. The corpus input is a TraceLab type that allows easy preprocessing, such as stop word removal, word splitting, and many others. The LSI Space builder computes TF/IDF implicitly, and it uses LAPACK's *dgesdd* [4] function to compute the Singular Value Decomposition (SVD) of the resulting matrix.

B. LSI Querier

This component is responsible for executing queries on a given LSI space, see Fig. *3*. It takes multiple inputs; the LSI space to query, the dictionary of document titles, the dictionary of vocabulary, and a set of queries to execute against the corpus. The queries input are in the same form as the corpus. Therefore, that preprocessing can be used also for the queries.

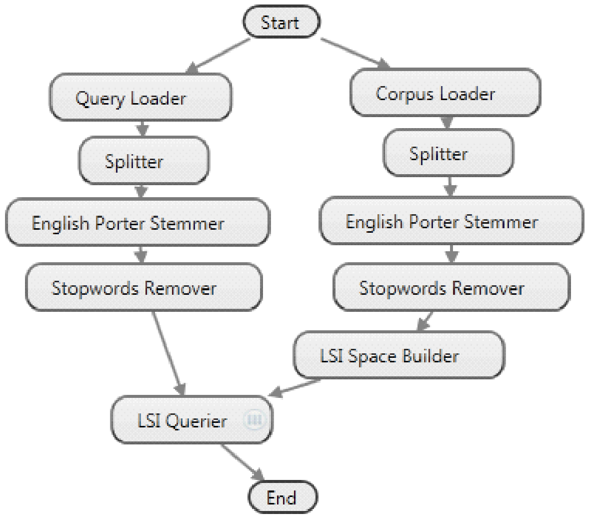


Fig. 1. An example of experiment set up of how to preprocess a loaded corpus and set of queries to the LSI Space Builder and LSI Querier respectively.

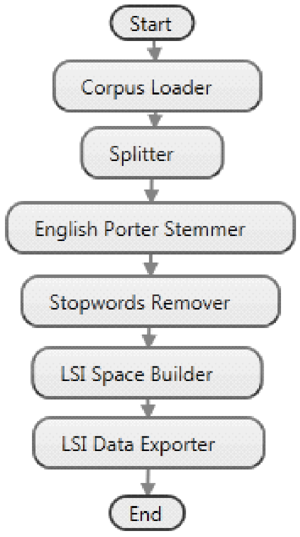


Fig. 2. An example of experiment set up of how to use the LSI Space Builder with the LSI Data Exporter.

C. LSI Data Importer

It imports the LSI space, the dictionary of document titles, and the dictionary of vocabulary from the file system. So the data can easily be reused. This allows for multiple different sets of experiments to be run on the same corpus without needing to rebuild the LSI space every time.

D. LSI Data Exporter

It exports resulting LSI data onto the file system for reusing with the LSI Querier. It takes the output from an LSI Space Builder as input and allows the LSI space, dictionary of document titles, and dictionary of vocabulary to be saved onto a specifiable location on the file system.

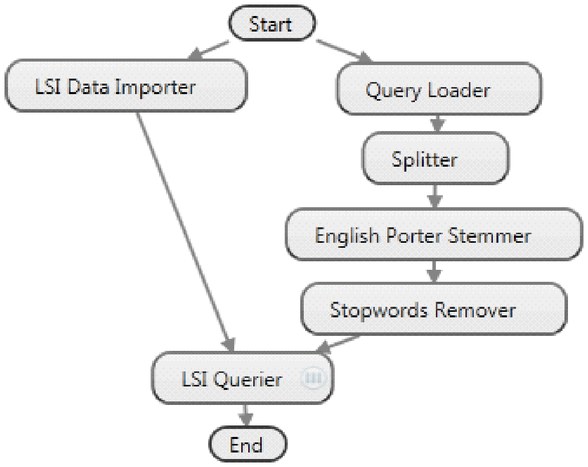


Fig. 3. An example of experiment set up of how to use LSI Querier, and LSI Data Importer to query the corpus, which was saved to the file system. The queries are preprocessed using the right side of the graph.

For our experiment, we used examples shown in Fig. *1* Fig. *3* to save, reload, and query the resulting LSI data multiple times. We created two corpora, one for the documentation and the other one for the function of the system. We then used the corpus consisting of the documentation to query the LSI space built from the function corpus, and vice versa. The ending result of the experiment is the traceability links between the external documentation and the functions of the system.

For future work on our components, we plan to create a Singular Value Decomposition algorithm, which directly interacts with the matrix types inside of TraceLab. We also plan to create a single component that, given two corpora and their LSI Spaces, will compute the traceability links and turn that for further analysis within TraceLab.

III. Traceability Link Data Set

Our goal is to uncover traceability between source code and other artifacts. This includes user documents (e.g., *HTML*, *XML/docbook*, *LaTeX* and *Doxygen*), build management documents (*automake*, *cmake*, and *makefile*), HowTo guides (e.g., *FAQs*), release and distribution documents (e.g., *ChangeLogs*, *whatsNew*, *README*, and *INSTALL* guides), progress monitoring documents (*TODO* and *STATUS*), and extensible mechanisms (e.g., *Python*, *Ruby*, and *Pearl* bindings for an *API*). These artifacts can be considered software informalisms [5].

We derive our traceability links by mining adaptive commits to uncover links caused by the adaptive maintenance process. The main goal of this is to help us assess the accuracy of using TraceLab in correctly uncovering the tractability links between source code files and other artifacts. Here, we analyze sets of files that frequently co-occur in adaptive change-sets by applying a frequent-pattern mining technique (i.e., sequential pattern mining).

The input data to frequent-pattern mining algorithms are in the form of transactions. A transaction refers to a group of commits that share a common property or occur in the same event (e.g., customer baskets or items checked out together in market-basket analysis) [6]. In this work, all the adaptive change-sets committed by a given committer within the same day are placed in a single group.

The number of groups is equal to the number of unique committers and day combinations. The ordered pattern found using this grouping implies that if a file is modified in a pattern by a committer the following or preceding files are likely to be modified by the same committer in the same day. The number of transactions in which a pattern occurs is known as its support. The support of a pattern is the number of groups in which it appears. Therefore, if the support of a pattern is at least a user-specified minimum support then it is a frequent pattern in the considered dataset. Sequential pattern mining produces a partially ordered list of files for patterns and as such, we term these ordered patterns.

To accomplish this uncovering process, we developed a sequential-pattern mining tool that is based on the Sequential Pattern Discovery Algorithm (SPADE) [7], which utilizes an efficient enumeration of ordered patterns based on common-prefix subsequences and division of search space using equivalence classes. Additionally, it utilizes a vertical input-transaction format (i.e., a set of transactions for each file vs. a set of transactions consisting of files) for efficiency.

Here, we examined the open source system KDE/Koffice as a case study. During this period of 1/1/2006 to 12/31/2010, there are nearly 36,000 commits (in subversion) to the KDE/Koffice project. Of these 36,000, we manually identified 131 as adaptive changes related to the porting of Qt 3 to Qt 4. These 131 commits were identified by searching through the commit log messages and manual verifying that they were indeed an adaptive change.

Mining adaptive commits of KDE/Koffice system uncovered 89 non-source code files, which have a traceability links at minimum support of three. Again, the traceability links identified will be used to validate “how well” the TraceLab discovers the existence of a traceability links between source code files and other artifacts. Therefore, the uncovered non-source code files that have traceability links will be used to generate queries in the evaluation process.

IV. THE TRACEABILITY RECOVERY PROCESS

The traceability recovery process we present centers on the LSI component [3]. However, user input is necessary and the degree of user involvement depends on the type of source code and the user’s task. Recovering the links between source code and documentation supports various software engineering tasks [8, 9]. Different tasks (and users) typically require different types of information. For example, there are times completeness is important. That is, the user needs to recover all the correct links even if that means recovering many incorrect ones at the same time. Other times, precision is preferred and the user restricts the search space so all the recovered links will be correct ones, even if this means not finding them all. Our TraceLab based solution tries to accommodate both needs (separately of course). One way to accommodate the user needs is by offering multiple ways to recovering the traceability links [10].

The traceability recovery process is organized in a pipelined architecture; the output from one phase constitutes the input for the next phase. However, TraceLab supports components to accomplish all of theses phases. The user’s involvement in the process occurs in the beginning for selecting the source code and documentation files. Then the user selects the dimensionality of the LSI subspace. After the LSI subspace is generated, the user determines what type of threshold will be used in determining the traceability links.

The input data consists of the source code and the external documentation. Our golden set are the tractability links uncovered in the previous section. In order to construct a corpus that suits LSI, a simple preprocessing of the input texts is required. Both the source and the documentation need to be broken up into the proper granularity to define the documents, which will be represented as vectors [3, 9]. Therefore, we split up the source code into documents of different granularity levels (i.e., functions, methods, interfaces, and classes). For external documentations, a paragraph is used as the granularity of a document. TABLE I contains the size of the system, as well as the dimensionality used for the LSI subspace and the determined vocabulary.

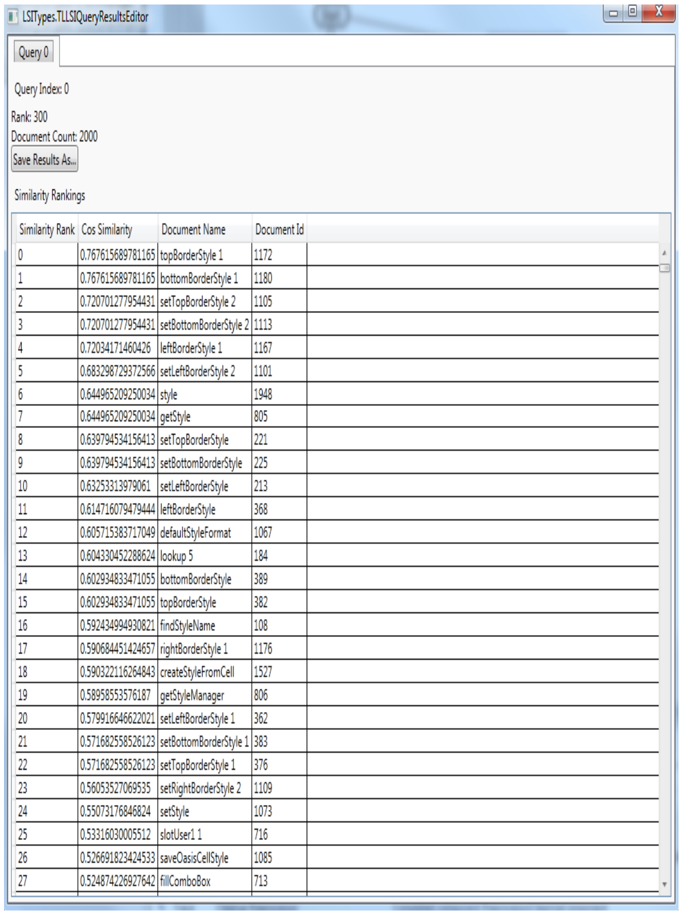
TABLE I . ELEMENTS OF THE KDE/KOFFICE SOURCE CODE, DOCUMENTATION AND LSI SETTINGS USED IN THE EXPERIMENTS

KDE/Koffice	Count	Documents
Source Code Files	1057	11492
Non-Source Code Files	89	102
Total # Documents		11594
Vocabulary	12839	-
LSI Dimensionality Used	300	-

Since the number of external documentations is much smaller than the number of source code files, we decided to trace the links from the external system documentations to the source code, rather than vice versa. Thus, a typical query will be used to find out which parts of the source code are described by a given external documentation. TABLE II summarizes the results we obtained on recovering the traceability links between external documentation and source code for KDE/Koffice. The first column (Cosine) represents the threshold value; column 2 (Total links retrieved) represents the total number of recovered links (correct + incorrect); and the last two columns are the precision and recall for each threshold. Comparing with the results in [9], TraceLab components enhance the precision and recall results. Fig. *4* shows a snapshot for the results of running one query sample over 2000 document.

TABLE II. RECOVERED LINKS, RECALL, AND PRECISION USING COSINE VALUE THRESHOLD FOR KDE/KOFFICE

Cosine threshold	Total Links Retrieved	Precision	Recall
0.60	184	40.76%	84.26%
0.65	133	51.87%	77.52%
0.70	95	57.89%	61.79%



Query Index: 0

Rank: 300

Document Count: 2000

Similarity Rankings

Similarity Rank	Cos Similarity	Document Name	Document Id
0	0.767615689781165	topBorderStyle 1	1172
1	0.767615689781165	bottomBorderStyle 1	1180
2	0.720701277954431	setTopBorderStyle 2	1105
3	0.720701277954431	setBottomBorderStyle 2	1113
4	0.72034171460426	leftBorderStyle 1	1167
5	0.683298729372566	setLeftBorderStyle 2	1101
6	0.644965209250034	style	1948
7	0.644965209250034	getStyle	805
8	0.639794534156413	setTopBorderStyle	221
9	0.639794534156413	setBottomBorderStyle	225
10	0.63253313979061	setLeftBorderStyle	213
11	0.614716079479444	leftBorderStyle	368
12	0.605715383717049	defaultStyleFormat	1067
13	0.604330452288624	lookup 5	184
14	0.602934833471055	bottomBorderStyle	389
15	0.602934833471055	topBorderStyle	382
16	0.592434994930821	findStyleName	108
17	0.590684451424657	rightBorderStyle 1	1176
18	0.590322116264843	createStyleFromCell	1527
19	0.58958553576187	getStyleManager	806
20	0.579916646622021	setLeftBorderStyle 1	362
21	0.571682558526123	setBottomBorderStyle 1	383
22	0.571682558526123	setTopBorderStyle 1	376
23	0.56053527069535	setRightBorderStyle 2	1109
24	0.55073176846824	setStyle	1073
25	0.53316030005512	slotUser1 1	716
26	0.526691823424533	saveOasisCellStyle	1085
27	0.524874226927642	fillComboBox	713

Fig. 4. Snapshot for the results of running one query sample (Results with first 2000 documents & LSI Dimensionality =300

V. Conclusion and Future Work

In this paper we proposed a TraceLab-based solution for conducting rapid experiments in traceability uncovering research. We provided the details for an environment that allows researchers to recover traceability links between external documentation and source code, using an information retrieval method, namely Latent Semantic Indexing (LSI). A set of experiments was presented and the results validated by comparing them with uncovered links extracted by applying a frequent-pattern mining technique on a set of adaptive commits of KDE/Koffice system.

The results are promising enough to demonstrate TraceLab as an environment that can be used to conduct traceability link uncovering research, and aid the growth in the traceability linking area.

In future, we plan to employ TraceLab to be used in supporting other software engineering researches, including classifying change commits, and predicting future maintenance activities.

Acknowledgment

This work was supported in part by grants from the National Science Foundation NSF grant CCF 08-11-21 and CNS 09-59924.

References

[1] E. Keenan, A. Czauderna, G. Leach, J. Cleland-Huang, Y. Shin, E. Moritz, M. Gethers, D. Poshyvanyk, J. Maletic, J. H. Hayes, A. Dekhtyar, D. Manukian, S. Hossein, and D. Hearn, "TraceLab: An experimental workbench for equipping researchers to innovate, synthesize, and comparatively evaluate traceability solutions", in Proceedings of the 34th International Conference on Software Engineering (ICSE), 2012, pp. 1375-1378.

[2] B. Dit, E. Moritz, and D. Poshyvanyk, "A TraceLab-based solution for creating, conducting, and sharing feature location experiments", in Proceedings of the 20th IEEE International Conference on Program Comprehension (ICPC), 2012, pp. 203-208.

[3] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis", Journal of the American Society for Information Science, vol. 41, 1990, pp. 391-407.

[4] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. J. Dongarra, J. D. Croz, S. Hammarling, A. Greenbaum, A. McKenney, and D. Sorensen, LAPACK Users' guide (third ed.): Society for Industrial and Applied Mathematics, 1999.

[5] W. Scacchi, "Understanding the requirements for developing open source software systems", IEE Proceedings-Software, vol. 149, no. 1, 2002, pp. 24-39.

[6] H. Kagdi, J. I. Maletic, and B. Sharif, "Mining Software Repositories for Traceability Links", in Proceedings of the 15th IEEE International Conference on Program Comprehension (ICPC), 2007, pp. 145-154.

[7] M. J. Zaki, "SPADE: An Efficient Algorithm for Mining Frequent Sequences", Machine Learning, vol. 42, no. 1-2, 2001, pp. 31-60.

[8] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, "Recovering traceability links between code and documentation", Journal of the IEEE Transactions on Software Engineering, vol. 28, 2002, pp. 970-983.

[9] A. Marcus and J. I. Maletic, "Recovering documentation-to-source-code traceability links using latent semantic indexing", in Proceedings of the 25th International Conference on Software Engineering (ICSE), 2003, pp. 125-135.

[10] M. Hammad, M. L. Collard, and J. I. Maletic, "Automatically identifying changes that impact code-to-design traceability during evolution", in Proceedings of the 17th IEEE International Conference on Program Comprehension (ICPC), 2009, pp. 35-64.