




Using Graph Embedding to Improve Requirements Traceability Recovery

Shiheng Wang, Tong Li^(✉) , and Zhen Yang

Beijing University of Technology, Beijing 100124, China
yeweimian21@163.com, {litong, yangzhen}@bjut.edu.cn

Abstract. Information retrieval (IR) is widely used in automatically requirements traceability recovery. Corresponding approaches are built based on textual similarity, that is, the higher the similarity, the higher possibility of artifacts related. A common work of many IR-based techniques is to remove false positive links in the candidate links to achieve higher accuracy. In fact, traceability links can be recovered by different kinds of information, not only the textual information. In our study, we propose to recover more traceability links by exploring both textual features and structural information. Specifically, we use combined IR techniques to process the textual information of the software artifacts, and extract the structural information from the source code, establishing corresponding code relationship graphs. We then incorporate such structural information into the traceability recovery analysis by using graph embedding. The results show that combined IR techniques and using graph embedding technology to process structural information can improve the recovery traceability.

Keywords: Requirements traceability recovery · Graph embedding · Structural information

1 Introduction

Software projects usually consist of many software artifacts, such as requirements documents, source code. The traceability recovery can get the relationships between these artifacts [5]. Traceability links is critical role for software comprehension. Usually, it need to create and maintain traceability links in whole software lifecycle. Unfortunately, in most cases, traceability links recovered from current artifacts. The creation and maintenance of traceability links are primarily manual. It consume lots of work and easy to make mistake. A completely manual traceability approach is usually only applicable to small projects. Although many researches attempt to automate the work, it has encountered a lot of difficulties due to the poor accuracy and too many false positives traceability links. Since most software artifacts contain textual data, many approaches are based on IR techniques. This kind of approach considered that the artifacts have high textual similarity are related. The source and target artifacts form candidate

links. The software engineer can discriminate the right or wrong links in the candidate links [2].

If the word used in the source artifacts are same to the word used in the target artifacts, IR-based techniques will have a good performance. But researchers often encounter the lexical mismatches problem due to artifacts developed by different software engineers. Therefore, IR-based approaches suffer from the lexical mismatches [1]. In addition, some artifacts usually are short, most IR techniques cannot get the suitable similarity values between them. This leads to many correct links fall to the end of the candidate list. These problems limit the traceability recovery.

Some approaches [1] focus on lexical transformations to solve problem. While others utilize different types of information to recover the links between artifacts [6], such as structural information of the code.

In our research, we propose to use textual information and structural information of software artifacts in combination to enhance the traceability recovery. We propose to utilize the structural information more effectively by using graph embedding.

Specifically, the contributions of our research are as follows:

1. Embedding code relationship graph by using the graph embedding technology, effectively expressing and utilizing the structural information of the code.
2. Analysis and extraction of various structural relationships such as inheritance, implement, parameters, and return values between the source code.
3. Comprehensive use of textual information and structural information to enhance the traceability recovery.

The paper is structured as follow. The second part discusses related work. The third part describes the background information of traceability recovery. The fourth part describes the approach we proposed. The fifth part describes the evaluation of the experiment. The sixth part analysis the effectiveness of the experiment. The seventh part summarizes the paper. The eighth part is discussion and future work.

2 Related Work

Information retrieval is a widely adopted technology in traceability recovery. [7] using the IR methods: vector space model (VSM), probabilistic Jensen and Shannon (JS) models, and relational topic modeling (RTM). [8] Statistically analysis of widely used IR methods: JS, VSM, Latent Semantic Index (LSI) and Latent Dirichlet Allocation (LDA). [4] proposes a simple method that only uses the nouns in the artifacts to improve the accuracy of the traceability recovery method. [5] proposed an automation technique that using machine learning techniques for LDA. The approach saves traceability links and learns probabilistic topic model. The learning model achieves semantic classification and visualization themes. Although [4-7] continues to improve the IR method and extract

more semantic information, it is one-sided to focus on textual information only, and the results in implementation are not ideal.

The structural information between source code is also very useful in traceability recovery. [3] introduced a way for traceability recovery in requirement document by using textual and structural information. [3] speculated that the relevant requirements share relevant source code elements. Building Evolving Interoperation Graph (EIG) using the JRipples [11] structural analysis tool. The traceability link graph (TLG) is proposed that encodes requirements and source code into nodes, and edges between nodes are recovery links. [1] proposed to use the textual information of the verb-object phrase in the requirement and source comments and the structural information of the source code to perform traceability link recovery. They process the natural language text, such as requirements and code comments, and source code separately. They use WordNet to handle synonym problems. [2] proposed to use feedback from software engineers to classify links to enhance the effect structural information. Specifically, they utilize structural information only when the engineer verifies the traceability link and classifies it as the correct link. Although the results of [1–3] have improved, structural information has not been fully explored.

There are other ways to come up. [6] proposed to use code ownership information to capture the relationship of source code. [9] proposed a neural network architecture that uses word embedding and RNN techniques to automatically generate traceability links. [10] proposed a traceability recovery method called Trustrace to identify traceability links by building VSM in the CVS/SVN change log. [12,13] proposed to use closeness analysis of code dependency. These methods improve the results, but limited.

3 Background

This section outlines the techniques involved in the traceability recovery process.

3.1 IR-Based Traceability Recovery Process

The IR-based traceability recovery firstly indexes the artifacts by extracting terms from their content. The text normalization phase is required before the indexing process. The output of the indexing process is an $m \times n$ matrix (term-by-document matrix). The widely used weighted mode is TF-IDF, which places more emphasis on words appear many times in the document and contain in minority documents, so it has high discriminative weight.

Engineers can use IR methods to compare artifacts after artifacts are indexed, and sort the similarities of all artifacts.

3.2 Latent Dirichlet Allocation

LDA is a generative probabilistic model. Each document has multinomial distribution over T topics, and each topic has multinomial distribution over the words of the corpus.

To compare the similarities of the documents, we use the Hellinger Distance, which is a measure of symmetric similarity between two probability distributions. Previous research on topical modeling also used it as a means of capturing similarities between documents. The Hellinger distance is defined as (1).

$$H(P, Q) = \sum_{x \in X} (\sqrt{P(x)} - \sqrt{Q(x)})^2 \quad (1)$$

The number of topics is key, and how to choose the correct number is undecided.

3.3 Graph Embedding

Information networks are common in society. Analysis of large information networks has attracted more and more attention from academia and industry. Graph embedding is a representation learning technique that maps nodes in a graph to real number vectors, and expresses the relationship in the graph by the relationship between the vectors.

The main purpose of Graph Embedding is to map the nodes to the vectors. These vectors are used to represent the relationship in the original graph.

This paper uses the LINE (Large-scale Information Network Embedding) [14]. After defining the objective function, LINE use a new edge-sampling which samples according to the probabilities of the edges weight. The first-order approximation can map directly connected nodes to closer distances, and the second-order approximation can map nodes with the same neighbor to a closer distance. Therefore, the first-order approximation can retain more local structural information, while the second-order approximation can retain more global structural information. As shown in Fig. 1, the node 6 and node 7 have a very high weight edge connection between them, and the node 5 and node 6 have many identical neighbor nodes. Considering the two approximations, the nodes 5, 6, and 7 are all Map to a closer distance in the embedding space.

4 Approach

4.1 Overall Process

The overall process is as shown in the Fig. 2. We extracting the textual information of the document and source code, and calculating the text similarity between the software artifacts. In our case, the document is use case. For the source code, we also analyses the structural information between the source codes besides extracting the textual information, organizes the source code into a graph, embeds it using the graph embedding algorithm, and the get the embedding vector of the code nodes, calculates the distance between the code nodes. The candidate links are generated based on the textual similarity and distance, and we set a threshold to filter the candidate links.

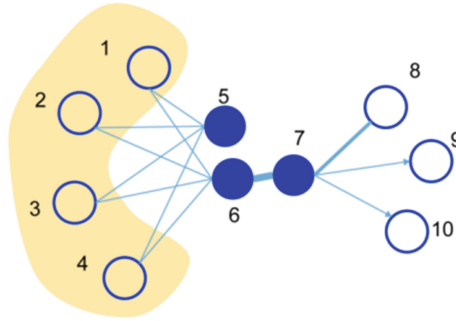


Fig. 1. An example of information network

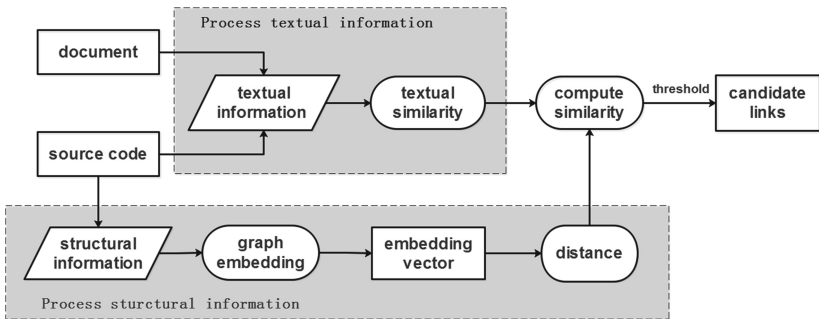


Fig. 2. The overall process

4.2 Process Textual Information

The software artifacts usually contain a large amount of semantic information, so the text information of the software artifacts can be extracted and processed by using the IR technology.

It need to preprocess the text of the software artifacts. All software artifacts need to: Segmentation of words: Dividing words according to separator. (ii) Morphological analysis: conversion of word case, conversion of nouns singular and plural, recovery of abbreviations, extraction of stems. (iii) Deletion of stop words, key words and most non-text marks.

The identifier of the source code, such as the class name, method name, contains important information, so we must handle the identifier appropriately. Identifiers often use Camel-Case and consist of several different words, distinguished by the case of the first letter of the word. So we need to split the source code identifier into separate words based on the letter case.

Then indexing the processed document. The $m \times n$ matrix (term-by-document matrix) will be gotten after the indexing process. The widely used weighted mode is TF-IDF, which places more emphasis on words often appear in the document and contain in minority documents.

Then the engineers can use IR methods to compare artifacts and rank the similarities.

Firstly, we use the TF-IDF and the LDA model to calculate the textual similarity, and the similarities of the artifacts are sorted to generate the candidate links. We use the use case and source code as source and target artifacts.

4.3 Process Structural Information

The Construction of the Code Graph. Source code artifacts not only contain textual information, but also inheritance and other structural relationships between source code, and such structural information between source code is very valuable for recovering traceability links of software artifacts. Therefore, we should not simply treat the source code as plain text, but rather to discover the structural relationship between the source code to help recovery traceability links of the software artifacts. The dataset software repository selected in this paper is developed by Java. Therefore, we propose the following six relationships between source code, namely inherit, implement, attribute, parameter, return and exception, and represented by the set $Relationship = \{Inherit, implement, attribute, parameter, return, exception\}$. The relationship is shown in Table 1.

Table 1. Relationship between source code

Relationship	Description	Weight
Inherit	Class A inherit class B	1
Implement	Class A implement interface B	1
Attribute	Class A has a class B type attribute	1
Parameter	Class A has a method whose argument type is Class B	1
Return	Class A has a method whose return type is Class B	1
Exception	Class A has a method whose exception type is Class B	1

We can organize the source code as a graph. The nodes in the graph are the source code classes, and the edges represent the relationships between the code elements. The graph structure is a natural and effective representation of the structural relationship of the source code. The code relational structure diagram can fully reflect the relationship between the source code. For the software project selected in this article, we can use existing tools to parse its source code.

There are many tools for parsing Java source code. These tools can parse Java software projects and extract code elements and the relationship between them. This article uses Eclipse JDT, a lightweight code analysis tool that can quickly builds Java code to a DOM structure's Abstract Syntax Tree (AST). Each element in the code corresponds to a node on the AST. By traversing the

AST, you can get all the code elements and the relationships between them to build a software code structure diagram.

We use Eclipse's JDT tool to parse the source code, get the structural information, and construct the corresponding relationship graph of the structural information.

The graph $G(C, E)$ a directed weighted graph that shows the relationship between code classes, where the vertex set $C = \{c_1, \dots, c_n\}$ is a collection of code classes, and the edge set $E = \{(c_i, c_j), \text{ the relationship between } c_i \text{ and } c_j\}$ is a collection of edges. The S is the collection of source artifacts. The $List = \{(s, c)\}$ is the candidate links.

In software projects, the closeness of relationship between different source code classes is different, and there are more relationship types and relationship numbers between source code classes with more closely related relationships. Therefore, we believe that if there are multiple relationship types or multiple relationships between two source code classes, the relationship between the two source code classes should be closer. Therefore, the edges (c_i, c_j) in the code graph $G(C, E)$ are weighted, and the weights represent the closeness of the relationship between the source code classes. The specific definition of the weight is as follows: for each relationship between two source code classes, the weight of the edge between them is incremented by one. For example, if there is one attribute relationship between the code nodes c_i and c_j , two parameter relationships, and two return value relationships, the weight between the edges (c_i, c_j) between them is 5.

Embedding the Code Graph. The main purpose of graph embedding is mapping the nodes to low dimension vectors. The structural information here can be different levels order. The first-order structural information can keep the distance between the adjacent nodes in the embedding space is still very close, and the higher-order structural information can keep the distance of the nodes with similar contexts in the embedding space still very close.

In this paper, we use LINE which can work for any type of information network: undirected, oriented, unweighted or weighted. This method optimizes the objective function and preserves the network structure. The algorithm is very efficient in practice.

We use the LINE graph embedding technology to embed the code nodes, and map them into low-dimensional space. If the two source code classes have a close relationship, the distance between the vectors of the two source code classes will be closer. Therefore, the closeness of relationship between the source codes can be expressed according to the distance between the vectors corresponding to the code nodes, that is, if the two source code classes have closer distances, the relationship between them is closer.

The reason why the graph is embedded in this paper is that when using the structural information between the codes to calculate the similarity between use case and source code, the distance between any two vertices on the graph needs to be calculated. By using the graph embedding technique to obtain the vector

corresponding to the source code node, the distance between the source code nodes can be calculated. There are many kinds of distances between two vectors in the representation space. In our study, the Euclidean distance (2) is used to represent the distance between two source code nodes.

$$d(x, y) := \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \dots + (x_n - y_n)^2} = \sqrt{\sum_{i=1}^n (x_i - y_i)^2} \quad (2)$$

4.4 Calculate the Similarity by Combining the Textual and Structural Information

We have calculated the textual similarity between artifacts and the distance between nodes, and then we will update the candidate links with textual information and structural information.

We proposed that if a text artifact d (use case) has a high similarity to the source code c_i , and the source code c_j is associated with the source code c_i in the structural information, then the similarity between d and c_j should be improved. On the contrary, if a textual artifact d (use case) has a low similarity to the source code c_i , and the source code c_j is associated with the source code c_i in the structural information, then the similarity between d and c_j should be reduced. Follow the same idea, we update the similarity between the text artifact d and the source code c_j according to the following formula (3), (4).

$$Sim(d, c_j) = Sim_{Text}(d, c_j) + \frac{Sim_{Text}(c_i, c_j)}{dist(c_i, c_j)} \quad (3)$$

$$Sim(d, c_j) = Sim_{Text}(d, c_j) - \frac{Sim_{Text}(c_i, c_j)}{dist(c_i, c_j)} \quad (4)$$

Finally, we will filter the generated candidate connections. There are usually two ways to do this. You can sort the generated candidate joins and pick the top k links with the highest similarity. Instead, we set a threshold for the candidate connection and filter out the links above the threshold.

5 Experiment Evaluation

In this section, we describe the evaluation the proposed method.

5.1 Dataset

The software repository of our research is eTour which is an electronic touristic guide. It contains 58 Use Cases, 174 classes and 366 correct links. The artifact language for the eTour systems is English. The correct traceability links which called oracle is restored to analyze the proposed experimental method.

5.2 Research Questions

In our research, we have proposed questions:

1. Whether combining the textual information and structural information of the software artifacts can improve the traceability link recovery work? Whether the structural information contributes to traceability links recovery work?
2. Whether use graph embedding technology to indicate the structural information of the source code can help the software artifact traceability recovery work?
3. Whether the combination of different IR methods can improve the traceability recovery method? Specifically, can a combination of different IR technologies achieve better accuracy and recall than using only one IR technology?

In response to our research questions, we compared only using the IR-based method with the combination of IR technology and graph embedding technology in the traceability recovery. For obtaining the experimental results of traceability recovery by combining different IR methods, we apply TF-IDF and LDA model combination to the recovery of artifact traceability links.

5.3 Metrics

Our research uses precision and recall metrics to evaluate proposed approach (5), (6).

$$precision = \frac{|correct \cap retrieved|}{|retrieved|} \% \quad (5)$$

$$recall = \frac{|correct \cap retrieved|}{|correct|} \% \quad (6)$$

We evaluate the result by comparing precision and recall.

5.4 Analysis of the Results

The Fig. 3 shows the number of correct links found in traceable link recovery. It can be seen from the figure that the combination of TFIDF and LDA, and using graph embedding technology for traceable link recovery can improve the number of correct connections captured, thus indicating that the combination of IR and graph embedding technology can improve traceability link recovery.

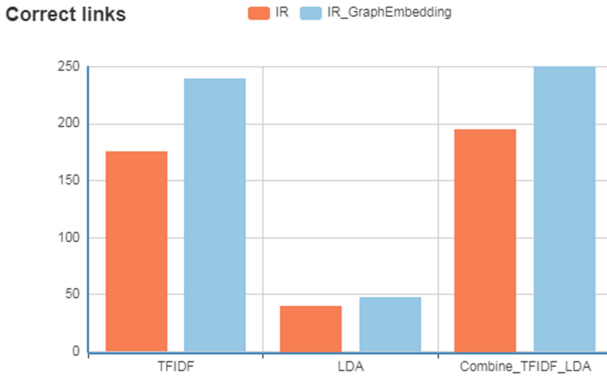


Fig. 3. The retrieved correct links

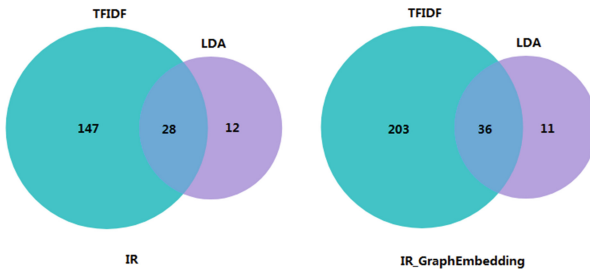


Fig. 4. The correct links captured by each method

The Fig. 4 shows the correct links captured by each method. Using graph embedding technology can capture more correct links. We also found that most of the correct links are captured by the TF-IDF method, but the LDA method also captures many correct links that ignored by TF-IDF.

The Fig. 5 shows the accuracy obtained in traceable link recovery. It can be seen from the figure that the use of combined IR method and Graph Embedding technology for traceable link recovery does not significantly improve accuracy, even with LDA technology, the accuracy is slightly reduced.

The Fig. 6 shows the recall obtained in traceable link recovery. It can be seen from the figure that using graph embedding technology for traceable link recovery can greatly improve the recall. In the case of using TFIDF and Combine TFIDF and LDA, the recall is significantly improved. In the LDA case, the recall has also increased slightly. This shows that graph embedding technology can significantly improve the recall of traceable links, and graph embedding helps traceability link recovery.

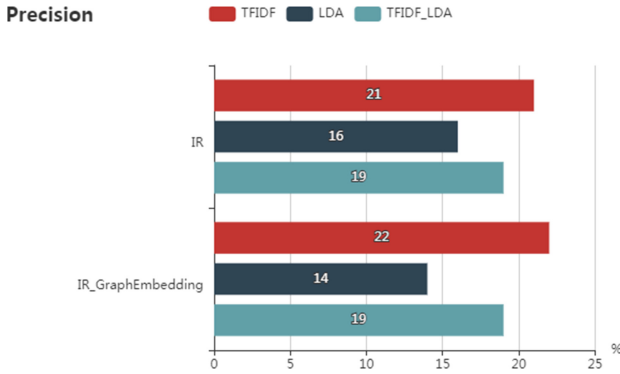


Fig. 5. The precision of traceability recovery

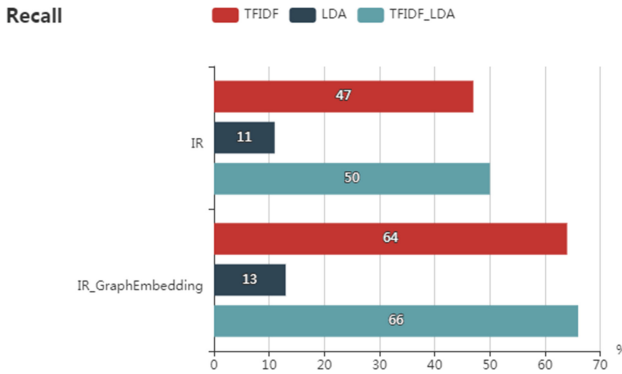


Fig. 6. The recall of traceability recovery

In conclusion, using graph embedding technology can capture more correct links and significantly increase the recall of traceability links, so graph embedding technology is helpful. And using multiple IR technologies can achieve better results than using one IR technology alone.

6 Threats to Validity

The repositories used in our research is difficult to compete with actual industrial projects. But it is near to the repository used in other research. ETour has been used as a benchmark repository for TEFSE 2011 traceability recovery challenges. Nonetheless, we plan to use other artifact repositories to replicate the experiment to confirm our findings.

Regarding the calculation of the distance between source code nodes, we use the Euclidean distance. Although Euclidean distance is useful, it has limitations. It treats the differences between the different properties of the sample as equivalent, which sometimes does not satisfy the actual situation. Therefore, the

Euclidean distance is applied to the case where the metrics of the components of the vector are unified.

We use LDA to calculate the text similarity of software artifacts. Although LDA is very useful, but it is very difficult to select the appropriate number of topics T . The inappropriate number of topics T will reduce the result.

7 Conclusion

Maintaining the traceability links often is an time-consuming work. The research strive for reducing manual work and increase productivity partly. The use of IR technique for recovering links has achieved promising results. But it often suffer the terms mismatch problem of artifacts developed by different people. In addition, some data have a lot of noise. In this case, this problem suppresses the IR technique to distinguish between related and unrelated artifacts. This article combines the textual information of the software artifacts with the structural information to recover the traceability link. Textual information of the software artifacts is processed by using different IR techniques, and the structural information of the source code is represented by using the graph embedding technology. We Improve the traceability recovery by comprehensively utilizing textual and structural information.

8 Discussion and Future Work

Our study uses the LINE model for graph embedding of source code nodes, which is applicable to any type of information network. In the future, we will use other graph embedding to embedding source code nodes.

Our study implementation on the eTour software repository. In the future we will try to recover traceable links for more software repository.

Our research focuses on the inheritance, implement, class attributes, function parameters, function return, and exceptions thrown by functions. These features are crucial for program comprehension and traceability recovery. We plan to utilize these and more features in other effective ways in the future.

Furthermore, the domain knowledge of software engineering is also significant for traceability recovery according to our empirical research. Therefore, we plan to utilize the logical reasoning to recover traceability links directly in the future.

Acknowledgement. This work is supported by National Key R&D Program of China (No. 2018 YFB0804703), International Research Cooperation Seed Fund of Beijing University of Technology (No. 2018B2), and Basic Research Funding of Beijing University of Technology (No. 040000546318516).

References

1. Zhang, Y., Wan, C., Jin, B.: An empirical study on recovering requirement-to-code links. In: 2016 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD). IEEE (2016)

2. Panichella, A., et al.: When and how using structural information to improve IR-based traceability recovery. In: European Conference on Software Maintenance and Reengineering. IEEE (2013)
3. Mcmillan, C., Poshyvanyk, D., Revelle, M.: Combining textual and structural analysis of software artifacts for traceability link recovery. In: Workshop on Traceability in Emerging Forms of Software Engineering. IEEE (2009)
4. Capobianco, G., De Lucia, A., Oliveto, R., Panichella, A., Panichella, S.: On the role of the nouns in IR-based traceability recovery. In: IEEE International Conference on Program Comprehension. IEEE (2009)
5. Asuncion, H.U., Asuncion, A.U., Taylor, R.N.: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - ICSE 2010 - Software Traceability with Topic Modeling, Cape Town, South Africa, 1–8 May 2010, vol. 1, p. 95. ACM Press (2010)
6. Diaz, D., Bavota, G., Marcus, A., Oliveto, R., Takahashi, S., Lucia, A.D.: Using code ownership to improve IR-based Traceability Link Recovery. In: IEEE International Conference on Program Comprehension. IEEE (2013)
7. Gethers, M., Oliveto, R., Poshyvanyk, D., Lucia, A.D.: On integrating orthogonal information retrieval methods to improve traceability recovery. In: IEEE International Conference on Software Maintenance, The College of William and Mary. IEEE (2011)
8. Oliveto, R., Gethers, M., Poshyvanyk, D., Lucia, A.D.: On the equivalence of information retrieval methods for automated traceability link recovery. In: 2010 IEEE 18th International Conference on Program Comprehension (ICPC). IEEE (2010)
9. Guo, J., Cheng, J., Cleland-Huang, J.: Semantically enhanced software traceability using deep learning techniques. In: 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE). IEEE Computer Society (2017)
10. Ali, N., Gueheneuc, Y.G., Antoniol, G.: Trust-based requirements traceability. In: 2011 IEEE 19th International Conference on Program Comprehension (ICPC). IEEE (2011)
11. Buckner, J., Buchta, J., Petrenko, M., Rajlich, V.: JRipples: a tool for program comprehension during incremental change. In: International Workshop on Program Comprehension. IEEE Computer Society (2005)
12. Kuang, H., Nie, J., Hu, H., Lü, J.: Improving automatic identification of outdated requirements by using closeness analysis based on source code changes. In: Zhang, L., Xu, C. (eds.) Software Engineering and Methodology for Emerging Domains. CCIS, vol. 675, pp. 52–67. Springer, Singapore (2016). https://doi.org/10.1007/978-981-10-3482-4_4
13. Kuang, H., Nie, J., Hu, H., Rempel, P., Lü, J., Egyed, A., Mäder, P.: Analyzing closeness of code dependencies for improving IR-based Traceability Recovery. In: 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER), February 2017, pp. 68–78. IEEE (2017)
14. Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., Mei, Q.: Line: large-scale information network embedding. In: Proceedings 24th International Conference on World Wide Web (WWW 2015), pp. 1067–1077 (2015)