Journal of Information Hiding and Multimedia Signal Processing ©2016 ISSN 2073-4212
Ubiquitous International Volume **7**, Number **1**, January **2016**

Towards Supporting Feature Location Using Syntactic Analysis

Jin-Shui Wang and Xing-Si Xue

College of Information Science and Engineering
Fujian University of Technology
Fuzhou 350108, China
wangjinshui@gmail.com; xxs@fjut.edu.cn

Shu-Chuan Chu

School of Computer Science, Engineering and Mathematics
Flinders University of South Australia
Australia
jan.chu@csem.flinders.edu.au

Received August, 2015; revised October, 2015

Abstract. *Feature location has been recognized as one of the most frequent and important activities undertaken by software developers. Aiming at the issue that most existing feature location approaches based on information retrieval are strongly affected by the quality of the documentation of software artifacts, this paper presents an improved IR-based feature location approach by syntactic analysis. In particular, the proposed approach firstly analyzes how terms have been used in the text through syntactic analysis. Then on this basis, the weight of these terms can be adjusted, and the key terms, which are able to describe the behavioral and semantic characteristics of software repositories, can be further extracted. The results of an experimental study conducted on two open source projects show that characterizing the context of software artifacts considering only key terms can effectively eliminate the text noise in software artifacts, and improve the accuracy of IR-based feature location methods.*
Keywords: Software maintenance; Feature location; syntactic analysis; Natural language processing; Term weighting.

1. **Introduction.** During software maintenance and evolution, developers often need to investigate the systems code base to locate and understand program elements (e.g., classes, methods) that are pertinent to a specific feature during software maintenance and evolution tasks. Such program comprehension activity is known as feature location (or concept location) in the context of software engineering [1, 2]. Feature location is one of the most common and important activities undertaken by software developers [3]. However, since various cross-cutting concerns of features often distributed in the source code and the complexity of software systems, the process of feature location is error-prone and time-consuming [4, 5].

Due to the essence of feature location is complex, in recent years, researchers have presented various techniques to provide automated assistance to help software developers accomplish feature location tasks. In terms of the applied technology, the proposed approaches can be categorized as following: Information Retrieval (IR) [6, 7], static analysis [8, 9], dynamic analysis [10], and the hybrid of several different techniques [11]. Since

IR is a low-cost and easy-to-use technique among these presented technologies, IR-based feature location techniques have been widely applied in practice and research.

The idea of IR-based feature location technique is that identifiers and comments encode domain knowledge, and a feature may be implemented using a similar set of terms (words), making it possible to find a features relevant code textually [3]. Unfortunately, no matter what IR technique being used, the quality of IR-based feature location techniques is heavily affected by the quality of the source code, i.e. conventions and/or the feature description [3]. In addition, for improving the maintainability and readability, most software documents (e.g., source code, requirement/feature document) inevitably contain some terms or phrases which are not related to its characteristics. These terms or phrases could become the noise that further affects the accuracy of IR-based feature location techniques. Moreover, apart from external documentation, the location and use of source code identifiers is the most important source of information in software maintenance [12]. Most exist feature location approaches model source code and features as bag-of-words, and each document is represented as a multi-set of words. In spite of the bag-of-words representation is easy to understand and calculate, it disregards all information about the order or syntactic structure of words. Although there are some differences between nature language, where nouns and verbs are the most crucial parts of a sentence [13], and software documents, nouns and verbs also play an important role in depicting the feature and function of source code [14, 15, 16]. In order to overcome these drawbacks, the main challenge is to precisely extract verbs and nouns as key terms from the artifact content to improve the performances of IR-based feature location techniques.

In this paper, we focus on improving IR-based feature location techniques using syntactic analysis. In particular, in order to extract the key terms, our approach first take source code and features as documents, and splits them into sentences. After that, a part-of-speech tagging is utilized to recognize the key terms within sentences, and then a chunk parsing is used to revise the errors that could be introduced in the process of part-of-speech tagging. Finally, an information retrieval technique, latent semantic indexing (LSI) is adopted to recover the link between source code and features. We conducted an experimental study on two open-source system to evaluate the effectiveness of our proposal. The results show with strong statistical significance that the recognized key terms can help in enhancing both precision and recall of feature location.

The remainder of the paper is organized as follows. Section 2 reviews related work. Section 3 describes the proposed approach. Section 4 presents the results of our experimental study and discusses some threats to our study. Section 5 concludes the paper with a summary of our findings.

2. **Related Work.**

2.1. **IR Methods used for Feature Location.**

Antoniol et al. [17] applied both a probabilistic model and a vector space model to trace C++ source code onto manual pages and Java code to functional requirements. The probabilistic model ranks feature (e.g., manual pages and functional requirements) according to the probability of being relevant to a source code unit. The vector space model treats documents and feature as vectors, and ranks documents against features by computing a distance function between the corresponding vectors. The results indicate that both probabilistic model and vector space model can provide a practicable solution to the problem of tracing links between source code and feature.

In order to extract the implicit semantics of the documentation and source code, Andrian and Jonathan [18] present a method to recover traceability links between documentation and source code using latent semantic indexing (LSI). In their study on using different IR method for recovering traceability between document and source code, they found that the method using LSI performs at least as well as other IR methods (e.g., probabilistic and VSM) with full parsing of the source code and morphological analysis of the documentation. They also observed that the method using LSI requires less processing of the source code and documentation, and less computation.

Lauren et al. [19] proposed a study of Latent Dirichlet Allocation (LDA) based feature location technique in with in which they measure the performance effects of using different configurations to retrieve 618 features from 6 open source Java systems. The results show that exclusion of comments and literals from the corpus lowers accuracy. Besides, they offered some specific recommendations for configuring the LDA based feature location technique.

Denys et al. [20] presented an empirical study to statistically analyze the equivalence of different IR-based traceability recovery methods. The comparison is based on Principal Component Analysis and on the analysis of the overlap of the set of the set of candidate links provides by each of the IR methods, e.g., Jensen-Shannon method, VSM, LSI, and LDA. The results show that while Jensen-Shannon, VSM, and LSI are almost equivalent and the accuracy of LDA is lower than previously used methods, LDA is able to capture some information missed by the other IR methods. It indicates that there is unlikely to be a clear winner among different IR-based traceability recovery or feature location methods. Therefore, developers can and should choose the appropriate IR method based on the specific context. However, most existing IR-based feature location methods focus on the retrieval process or the search result display, few studies have been concerned with text noise reduction and keywords extraction [15].

In general, an IR-based feature location process indexes all source code and features by extracting information about that occurrences of terms within them. However, terms might play different roles in capturing the semantics of the artifact content. Therefore, key terms should be identified and be weighted more heavily than others, as they can be regarded as more meaningful in identifying links between feature and source code. In this paper we assume that the most crucial terms in software artifact are nouns and verbs, since they are able to depict the feature and function of source code. Therefore, we propose to act on the artifact indexing process taking into account only the nouns and verbs contained in the contents.

2.2. **Syntactic Analysis.** Syntactic analysis is the key and difficult issue in the Natural Language Processing (NLP). In order to reduce the complexity of the full parsing, syntactic analysis is divided into several more manageable subtasks based on the idea of divided and conquer. Separating the Part-Of-Speech (POS) tagging and chunking from syntactic analysis is a successful strategy. Highly accurate and efficient POS taggers and chunk taggers are freely available [21], and thus the foundation of full syntactic parsing is built.

In any language, verbs and nouns are considered as the most important and basis parts of a sentence [13]. Verbs often play a connection role and describe the sentences action [21], and nouns characterize the semantics [15]. To fully express a specific meaning requires both verbs and nouns working together. Therefore, it is important to consider both verbs and nouns when analyzing sentences.

Hill et al. [21] found that using natural language information embedded in software artifacts can significantly improve the effectiveness of various software maintenance tools.

To leverage this information about programs, they proposed a set of rules for extracting verb-direct object (verb-DO) pairs from program method signatures. In a programming language, verbs or verb phrases usually appear in the identifiers of method names, and the identifier should occur to specify the name by which the method will be known. Therefore, the extraction of verb-DO information from method signatures is especially helpful in improving tools for comprehension and maintenance of object-oriented source code. However, the verb-DO pairs may not apply to other natural language information (e.g., comments) in source code. Firstly, there are usually more than one verb and noun in a sentence, making it difficult to identify the most suitable verb-DO pairs. Secondly, the extracted verb-DO pairs are often missing the subject as the kernel of a sentence. Therefore, the usefulness of the verb-DO pairs from comments is an open question [21].

Capobianco et al. [15] believed that the words that provide more indication on the semantics of a document are the nouns. According to that, they proposed to act on the software artifact indexing process taking into account only the nouns contained in the contents. Unfortunately, their proposed approach neglects the function of verbs to the sentences. For example, here is a feature description about auto-save in JEdit, if the user writes to disk, they want the changes. In this sentence, the set of noun is user, disk, they, changes, and the set of verb and noun is user, writes, disk, they, want, changes. By the contrast of these two sets, it is easy to find that the latter can preserve a more complete expression of the feature.

Shepherd et al. [22] defined an Action-Oriented Identifier Graph (AOIG) to reconnect the scattered actions in an OOP system. To extract an AOIG from source code, they performed various Natural Language Processing (NLP) techniques (i.e. POS tagging, chunking) to process the natural language clues left by programmers in source code and comments. However, they only anticipated the useful future for the AOIG rather than analyzed quantitatively the roles NLP techniques play in assisting perform feature location tasks.

3. **Approach.**

In this section, we first give an overview of our proposal in Section 3.1, and then the details of the implementation are given in Section 3.2.

3.1. **Overview.**

To implement syntactic analysis based feature location, our approach employs a POS tagger and chunking recognition to identify the nouns and verbs within software artifacts. Figure 1 presents an overview of our approach, showing the main steps and input/output of the approach.

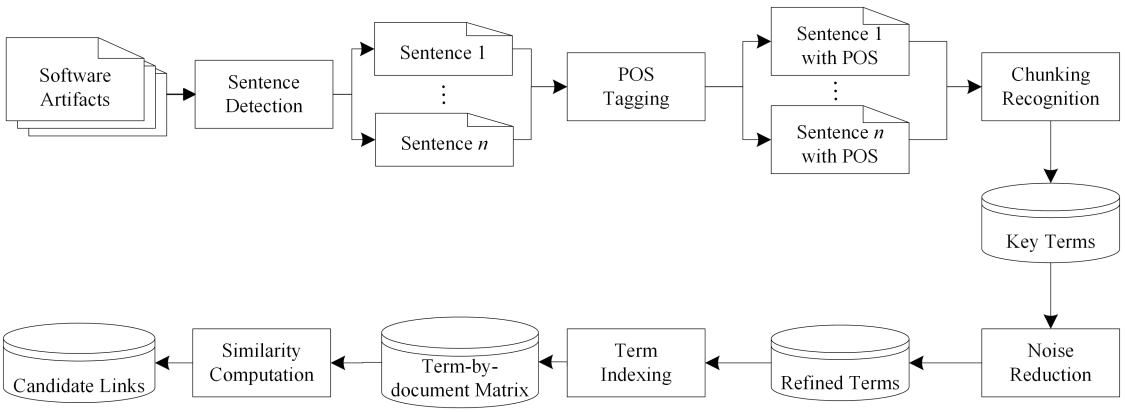

FIGURE 1. Overview of our approach

As shown in Figure 1, the artifacts are first split into sentences by a sentence detector which can detect whether a punctuation character marks the end of a sentence or not. After that, a POS tagging is performed to marks terms (words) with their corresponding type (e.g., verb, adjective) based on the term itself and the context of it. However, since a term might have multiple POS tags, a chunking is further carried out to reduce the errors which might be introduced during POS tagging. Aided by analyzing result of POS tagging and chunking, verbs and nouns could be identified and then marked as key terms to describe the behavioral and semantic characteristics of artifacts. As there may be some useless terms in various types of artifacts content, e.g., reserved word, which are not related to the characters of the contents might impact the performance of our approach, we applied a noise reduction to eliminate them and reduce such influence to a minimum. Based on these refined terms, all the artifacts in the repository can be indexed by extracting information about the occurrences of terms with in them. Finally, our approach ranks the textual similarity of all possible pairs of feature and source code. Pairs having a similarity above a certain threshold or being in the topmost positions of ranked list are regarded as candidate links.

In our work, the OpenNLP toolkit is utilized to accomplish the POS tagging and chunking, and the LSI over VSM is selected to compute the similarity. Specifically, the reason we choose LSI are as follows: (1) LSI can solve the problems of polysemy and synonymy well, which is important with respect to feature location problem because different developers may have their own favorite choices of words in the use of language; (2) Recent work has revealed that LSI outperforms VSM and Bayes classifiers [23].

3.2. **Overview.** There are six major steps when implementing syntactic analysis based feature location process, i.e. sentence detection, POS tagging, chunking recognition, noise reduction, term indexing, and similarity computation.

3.2.1. *Sentence Detection.* Since sentences are necessary input for syntactic analysis, the text in software artifacts need be segmented into its sentences ahead of time. However, there is no sentence in source code in a strict sense except for comments, and different feature location tasks may require different levels of program element granularity (e.g., classes, methods). Therefore, source code should be segmented according to the required levels of program element granularity.

First of all, if developers are concerned about the mapping relation of methods and artifacts, the source code should be segmented into a set of methods, and then we can take method comment as independent sentences, and append them onto the end of corresponding method. After that, all fields and comments which do not belong to any method will be discarded. Second, if developers are more interested in the mapping relation of classes and artifacts, the source code should be segmented into a set of classes, and then we can take class comment as independent sentences, and append them onto the end of corresponding class. Nevertheless, if the method body contains anonymous class in some programming languages such as Java, the anonymous class will be regarded as part of the method body. For instance, as can be seen from Figure 2, propertyChange is a method and it appears in an anonymous class, which locates in body of method FocusWindowAction, the method propertyChange is taken as part of the method FocusWindowAction, and will not be treated as an independent and complete analytic target.

3.2.2. *POS Tagging.* After segmenting the content of artifacts and source code into sentences, the types of terms (words) in those sentences should be marked by a POS tagger. However, since a term might have multiple POS tags, and existing POS tagging

FIGURE 2. Code Snippet Extracted from the JHotDraw Repository

methods cannot guarantee that all marked terms be absolutely correct. Take the sentence "Given a sorted linked list, delete all duplicates such that each element appear only once" as an example. After POS tagging, it will be marked as "Given$_{<vbg>}$ a$_{<dt>}$ sorted$_{<jj>}$ linked$_{<vbn>}$ list$_{<nn>}$, delete$_{<vbn>}$ all$_{<dt>}$ duplicates$_{<nns>}$ such$_{<jj>}$ that$_{<in>}$ each$_{<dt>}$ element$_{<nn>}$ appear$_{<vbp>}$ only$_{<rb>}$ once$_{<rb>}$." It can be inferred that although the term "sorted" is correctly marked as an adjective, the term "linked" is incorrectly marked as a verb in simple past.

Even the grammatical features of source code are different from natural language documents, the part of speech of terms in source code are still identifiable. Take the source code shown in Figure 2 as example. The statement "setEnabled(view != null)" will be marked as "setEnabled$_{<vbn>}$(view$_{<nn>}$!=null". It can be inferred that the method call "setEnabled" is marked as a verb, and the variable "view" is marked as a noun.

3.2.3. *Chunking Recognition.* Chunking recognition, also known as shallow parsing or partial parsing, is used to identify the constituents of a sentence and generate partial (shallow) analysis of sentences rather than a full parse. Although the result of chunking is not an entire syntactic tree, each chunker is a subgraph of the entire syntactic tree. With the complement of the attachment relationship between chunks, they can reduce the errors which might be introduced during POS tagging.

Let's take the sentence "*Given a sorted linked list, delete all duplicates such that each element appear only once*" as an example to show how chunking recognition collaborate with POS tagging. It will be divided into several chunks and then be marked as "Given$_{<vp>}$ [a sorted linked list]$_{<np>}$, delete$_{<vp>}$ [all duplicates]$_{<np>}$ such$_{<vp>}$ that$_{<sbar>}$ [each element]$_{<np>}$ appear$_{<vp>}$ [only once]$_{<advp>}$" after chunking. It can be inferred that "[a sorted linked list]" is grouped together into a noun phrase. According to the rules of English grammar, the word "linked" is correctly marked as an adjective. However, there are usually many proper nouns and terminology in source code and potentially affect of the accuracy of chunking recognition and POS tagging. More seriously, sentences within source code and other software artifacts do not always square with the grammatical rules. Let's take the feature numbered 1608948 within JEdit contains the

following sentence:"*The function protected void processFocusEvent(FocusEvent e) in HistoryComboBoxEditor is where the problem is*" as example. Although the sentence is not complex and easy to understand, it is difficult to assign part-of-speech tags to terms within this sentence. In allusion to the problems mentioned above, only the terms that are recognized as nouns or verbs by both POS tagging and chunking recognition will be preserved and considered as key terms and be used to describe the behavioral and semantic characteristics of artifacts.

In the sentence "Given a sorted linked list, delete all duplicates such that each element appear only once", the words "Given", "delete" and "appear" will be recognized as verbs, and the words "list", "duplicates" and "element" will be recognized as nouns, while others will be regarded as non-critical words and discarded. For example, the word linked is recognized as a verb by POS tagging, and then it is recognized as an adjective by chunking recognition. Because the tag results of the word are different, the word linked will be discarded and not involved in subsequent processing. Based on the tag results, the set of key terms, i.e. Given, list, delete, duplicates, element, appear will be identified from the sentence. By contrast analyses between the set of key terms and the sentence, it can be seen that the set of key terms can effectively filter out less important words, as well as preserve critical information of the original sentence.

As the above example shows, both verbs and nouns play have an important role to play in depicting the semantics of a software artifact. Therefore, they should be extracted and be identified as key terms to support IR-based feature location. Besides, Rather than relying solely on POS tagging, chunking recognition should be performed to tag terms part of speech more correctly.

3.2.4. *Noise Reduction.* As software artifacts (especially the source code) often contain noise (e.g., reserved word) which is unimportant and independent of the characters of themselves. To reduce the impact of noise on key terms, noise reduction is necessary. Given the identified candidate key terms within software artifacts, our approach first filters out reserved words in source code and stop words in other types of software artifacts, and then the candidate terms will be further refined by tokenization and word stemming.

Since the syntactic structure of sentences is probably influenced by noise reduction, which could affect the accuracy of syntactic analysis, noise reduction must be performed after POS tagging and chunking recognition. For instance, if noise reduction is performed firstly, the words sorted and linked within the sentence Given a sorted linked list, delete all duplicates such that each element appear only once will be respectively reduced as sort and link, which will be identified as two verbs in the following analysis process. As a result, these two words will be incorrectly preserved as key terms, which reduce the accuracy of syntactic analysis.

3.2.5. *Term Indexing.* To apply IR for feature location, a term-by-document matrix should be constructed where the row, column and cell refer to software artifacts (i.e., features and methods), terms, and the frequency of a term in an artifact, respectively. In order to makes full use of the natural language information embedded in software artifacts, in our work, a modified term indexing process is proposed. In specific, the term indexer builds the term-by-document matrix considering only the terms that are identified as refined key terms. Then, a grammatical-semantic matrix is proposed, which contains only key terms and removes all other terms that play a minor role. Formally, let $D = \{d_1, ..., d_n\}$ be the set of software artifacts, where n is the number of software artifacts, $T = \{term_i : tag(term_i) is key\_terms, i = 1...m\}$ be the set of key terms extracted from D. Given a feature location task consisting of a set of software artifacts, the term-by-document matrix

M can be build as follows: $M = \{t_{1,1}, ..., t_{i,j}, ..., t_{m,n}, 1 <= i <= m, 1 <= j <= n\}$, where $t_{i,j}$ is the frequency of the i^{th} key term in the j^{th} document.

3.2.6. *Similarity Computation.* Although the term-by-document matrix is easy to obtain and the similarity score between two vectors is also easy to compute, it is usually high-dimensional and sparse. Therefore, processing of such data requires lots of computing resources. Additionally, as some words could have multiple meanings, the term-by-document suffers from polysemy and homonym. In view of the above problems, we introduce Singular Value Decomposition (SVD) [24] which can be used to deriving a set of uncorrelated indexing variables or factors, and each word and document is represented by its vector of factor values. By replacing individual words with derived orthogonal factor values, the SVD representation can assist in solving the problems (e.g., high-dimensional, polysemy and homonym) of the term-by-document matrix. SVD is applied by LSI which is primarily utilized to match the concepts to project the origin term-by-document matrix into reduced k-space (where k is usually smaller than m). On this basis, LSI can further rank the similarity of all possible pairs of feature and source code, and then pairs being in the topmost positions of ranked list are regarded as candidate links.

4. **Evaluation.** In this section, an experimental study was conducted to evaluate whether our approach can improve developers feature location practice by extracting key terms in software artifacts. The tasks used in our study are selected from the benchmarks provided in the literature [3].

4.1. **Subject Systems.** Our study involves two open source Java systems, i.e. JEdit and MuCommander. JEdit is a well-known text editor with hundreds of person-years of development behind it, which contains 86 bug-related, 34 feature-related, and 30 patch-related features in our study. MuCommander is a lightweight, cross-platform file manager, which contains 81 defect-related and 11 enhancement-related features in our study.

To acquire all the artifacts relevant to the feature location tasks, we develop a simple software analyzing tool to extract methods by parsing abstract syntax tree of the specific version of system, and take all related source code description from different issue tracking systems as features. According to the type of system, features could be classified into five types: bug-related, defect-related, enhancement-related, feature-related and patch-related. Table 1 summarizes the subject systems, the version of system used to extract methods, and the features.

Table 1. Subject systems,version and features

System	Version	Period of SVN Commits Analyzed		#Feature Description
		Begin	End	
JEdit	4.3	4.2	4.3	150
MuCommander	0.8.5	0.8.0	0.8.5	92

4.2. **Methodology.** Our objective is to improving feature location practice by indexing only the key terms which are able to characterize the context of software artifacts. The rationale behind the proposal approach is that nouns and verbs play an important role in depicting the semantics of a software artifact, and the study aims at addressing the following research question:*does the proposed indexing process improve the accuracy of IR-based feature location methods?*

To answer this question we performed feature location tasks between methods and features of the subject systems using four different artifact indexing processes:

All: all the terms (e.g. adjectives, nouns) contained in the artifact are equally considered during the indexing process.
Verb: only verbs contained in the artifact are considered during the indexing process;
Noun: only nouns contained in the artifact are considered during the indexing process;
KeyTerm: only key terms identified by the proposal approach contained in the artifact are considered during the indexing process;

4.3. **Measures.** We evaluated the performance in feature location tasks by precision, recall and F-measure. Precision is the percentage of correctly reported links between features and methods. Recall is the percentage of actual links between features and methods that are reported. Given a set of feature location tasks T consisting of a set of features F, we computed the overall precision P_T and recall R_T for T as follows:$P_T = \sum_{f \in F} P_f/|F|, R_T = \sum_{f \in F} R_f/|F|$. F-measure for a feature location task set T is then computed as $f_b = (1+b^2)/(1/P_T+b^2/R_T)$ to reflect a weighted average of the precision and recall. In our study, following the treatment in [4], we set b to 2, i.e., recall is considered four times as important as precision, because finding missing links is more difficult than removing incorrect links.

TABLE 2. The quality(recall%, precision%, and f-measure%) of each feature location results achieved with different artifact indexing processes of different cut-point

	JEdit												MuCommander											
cut-point	10			20			50			70			10			20			50			70		
	r	p	fm	r	p	fm	r	p	fm	r	p	fm	r	p	fm	r	p	fm	r	p	fm	r	p	fm
all	30	3	10	51	2	10	82	1	7	93	1	5	12	2	6	24	2	7	57	2	8	76	2	7
noun	38	3	13	54	2	10	83	2	7	94	1	5	14	2	7	31	2	9	64	2	9	82	2	7
verb	17	1	6	26	1	5	58	1	5	79	1	5	16	2	7	28	2	8	53	2	7	71	1	6
key term	37	3	12	54	2	10	86	2	7	96	1	6	15	2	7	32	2	9	64	2	8	83	2	8

4.4. **Results.** Let us first investigate the quality (in terms of precision, recall and F-measure) of each feature location results achieved with the four artifact indexing processes (i.e., All, Verb, Noun, and KeyTerm) of different cut-point values in Table 2. Every artifact indexing process recovers only the top u links in the ranked list regardless of the values of the similarity measure when the cutpoint is is u. For the purposes of reproducibility and independent study, we have made all data available via an online appendix: http://dwz.cn/1xtYwG.

From the results, it can be seen that the precision is rather high, while the recall is very low (less than 10%). Through analysis of this subject systems and feature location tasks, there are two possible contributing factors we found. Firstly, the set of candidate links is composed of all pairs of features and methods, and the large masses of methods lead to a rapid expansion of candidate links. Take JEdit as example, The JEdit 4.3 used in this study consists of 7085 methods and 150 features, all of them can be combined to produce $7085 \times 150 = 1062750$ candidate links. Secondly, the number of correct links is rather small. For example, there are only 748 methods that is associated with 150 features, and 1466 links between them are correct. In other words, there are only 1466 correct links in 1062750 candidates, the proportion is just about 0.1%.

TABLE 3. Results of T-Tests of Hypotheses, for the variable precision, recall, and F-measure. Measurements are reported in the following columns: minimum value, maximum value, median, means (μ), variance (σ^2), the Degrees of freedom (DF), the Pearson correlation coefficient (PC), statistical significance (p), T_{crit}, and the T statistics.

H	Var	Approach	Samples	Min	Max	Median	μ	σ^2	DF	PC	T	T_{crit}	p	Decision
H0	*recall*	Verb	200	0.008	1	0.561	0.542	0.067	199	0.941	-20.755	1.972	< 0.001	Reject
		KeyTerm	200	0.028	1	0.755	0.676	0.072	199					
	precision	Verb	200	0.009	0.031	0.014	0.014	0	199	0.631	-10.218	1.972	< 0.001	Reject
		KeyTerm	200	0.009	0.079	0.018	0.019	0	199					
	Fmeasure	Verb	200	0.009	0.085	0.056	0.059	0	199	0.399	-13.856	1.972	< 0.001	Reject
		KeyTerm	200	0.030	0.135	0.079	0.077	0	199					
H1	*recall*	Noun	200	0.021	1	0.737	0.669	0.068	199	0.998	-6.170	1.972	< 0.001	Reject
		KeyTerm	200	0.028	1	0.755	0.676	0.072	199					
	precision	Noun	200	0.009	0.079	0.017	0.019	0	199	0.991	-2.947	1.972	0.004	Reject
		KeyTerm	200	0.009	0.079	0.018	0.019	0	199					
	Fmeasure	Noun	200	0.022	0.138	0.077	0.077	0	199	0.993	-3.055	1.972	0.003	Reject
		KeyTerm	200	0.030	0.135	0.079	0.077	0	199					
H2	*recall*	All	200	0.035	1	0.711	0.636	0.077	199	0.996	-21.839	1.972	< 0.001	Reject
		KeyTerm	200	0.028	1	0.755	0.676	0.072	199					
	precision	All	200	0.009	0.062	0.016	0.017	0	199	0.969	-9.958	1.972	< 0.001	Reject
		KeyTerm	200	0.009	0.079	0.018	0.019	0	199					
	Fmeasure	All	200	0.036	0.111	0.070	0.070	0	199	0.954	-14.131	1.972	< 0.001	Reject
		KeyTerm	200	0.030	0.135	0.079	0.077	0	199					

In most case (see Table 2), the proposed approach is better than other traditional indexing process, especially for increasing recall values (see our online report for more details). In order to conduct an overall test, their performance of different cutpoint values over the range of from 0.01 to 1 were recorded and compared. To evaluate the performance improvement of the proposed approach, we compare the precision, recall and F-measure of four feature location results. We introduce the following null hypotheses to evaluate how different the performance of them.

H0: There is no difference between the performance of feature location using artifact indexing processes KeyTerm and Verb.

H1: There is no difference between the performance of feature location using artifact indexing processes KeyTerm and Noun.

H2: There is no difference between the performance of feature location using artifact indexing processes KeyTerm and All.

We use paired sample t-tests to evaluate the null hypotheses H0, H1 and H2 in terms of precision, recall and F-measure. We evaluate the hypotheses at a 0.05 level of significance.

The result of these four tests are shown in Table 3. Based on the results we reject the null hypothesis H0, H1, and H2 for all the measures of precision, recall and F-measure. Therefore, we accept the alternative their alternative hypothesis, i.e., there is significant different between the performance of feature location using artifact indexing processes KeyTerm and others.

5. **Conclusions.** In this paper, we have presented an improved IR-based feature location approach using syntactic analysis. Since both nouns and verbs play an important role in depicting the semantics of a software artifact, in this work, they are extracted from artifacts content and identified as key terms, and then utilized to calculate the textual similarity between two artifacts.

In particular, source code and features as documents are first split into sentences. After that, a part-of-speech tagging is utilized to recognize the key terms within sentences, and then a chunk parsing is used to revise the errors that could be introduced in the process of part-of-speech tagging. Finally, an information retrieval technique, i.e. latent semantic indexing, is adopted to recover the link between source code and features based on the identified key terms. The results achieved in an experimental study on two open-source system demonstrated that our proposal is able to significantly improve the accuracy of an IR-based feature location using LSI.

In the future work, we will focus on integrating other mature natural language processing technology. We also plan to further the proposed artifact indexing approach by combining our multi-faced feature location tool *MFIE* [25] to provide a comprehensive support for searching, exploration, and recommendation in an interactive feature location process.

Acknowledgment. This work is supported by National Natural Science Foundation of China under Grant No. 61402108, Foundation for Scientific Research of Fujian Education Committee under Grant No.JA15348, JA14221, JB12146, Natural Science Foundation of Fujian Province of China under Grant No.JK2012033.

REFERENCES

[1] T. J. Biggerstaff, B. G. Mitbander, and D. Webster, The concept assignment problem in program understanding, *Proc. of the 15th Int'l Conf. on Software Engineering*, Maryland, USA, pp.482–498, 1993.

[2] V. Rajlich, N. Wilde, The role of concepts in program comprehension, *Proc. of 10th Int'l Workshop on Program Comprehension*, Paris, France, pp.271–278, 2002.

[3] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk, Feature location in source code: a taxonomy and survey, *Journal of Software: Evolution and Process*, vol.25, no.1, pp.53–95, 2013.

[4] J. H. Hayes, A. Dekhtyar, and S. K. Sundaram, Advancing candidate link generation for requirements tracing: the study of methods, *IEEE Trans. on Software Engineering*, vol.32, no.1, pp.4–19, 2006.

[5] A. J. Ko, B. A. Myers, M. J. Coblenz, and H. H. Aung, An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks, *IEEE Trans. on Software Engineering*, vol.32, no.12, pp.971–987, 2006.

[6] N. Alhindawi, N. Dragan, M. L. Collard, and J. I. Maletic, Improving feature location by enhancing source code with stereotypes, *Proc. of the 29th Int'l Conf. on Software Maintenance*, Eindhoven, Netherlands, pp.300–309, 2013.

[7] E. Hill, B. Sisman, and A. Kak, On the use of positional proximity in ir-based feature location, *IEEE Conf. on Software Maintenance, Reengineering and Reverse Engineering*, Antwerp, Belgium, pp.318–322, 2014.

[8] X. Peng, Z. Xing, X. Tan, W. Zhao, Improving feature location using structural similarity and iterative graph mapping, *Journal of Systems and Software*, vol.86, no.3, pp.664–676, 2013.

[9] M. Petrenko, V. Rajlich, Concept location using program dependencies and information retrieval (Depir), *Information and Software Technology*, vol.55, no.4, pp.651–659, 2013.

[10] C. Ziftci, I. Kruger, Feature location using data mining on existing test-cases, *Proc. of the 19th Working Conf. on Reverse Engineering*, Kingston, Canada, pp.155–164, 2012.
[11] B. Dit, M. Revelle, and D. Poshyvanyk, Integrating information retrieval, execution and link analysis algorithms to improve feature location in software, *Empirical Software Engineering*, vol.18, no.2, pp.277–309, 2013.
[12] J. Paakki, A. Salminen, and J. Koskinen, Hypertext support for the information needs of software maintainers, *Journal of Software Maintenance and Evolution: Research and Practice*, vol.16, no.3, pp.187–215, 2004.
[13] N. Saharia, U. Sharma, and J. Kalita, A suffix-based noun and verb classifier for an inflectional language, *Proc. of 2010 Int'l Conf. on Asian Language Processing*, Heilongjiang, China, pp.19–22, 2010.
[14] Y. Hayase, Y. Kashima, Y. Manabe, and K. Inoue, Building domain specific dictionaries of verb-object relation from source code, *Proc. of 15th European Conf. on Software Maintenance and Reengineering*, Oldenburg, Germany, pp.93–100, 2011.
[15] G. Capobianco, A. D. Lucia, R. Oliveto, A. Panichella, and S. Panichella, Improving ir-based traceability recovery via nounbased indexing of software artifacts, *Journal of Software: Evolution and Process*, vol.25, no.7, pp.743–762, 2013.
[16] S. Zamani, S. P. Lee, R. Shokripour, and J. Anvik, A noun-based approach to feature location using time-aware term-weighting, *Information and Software Technology*, vol.56, no.8, pp.991–1011, 2014.
[17] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, Recovering traceability links between code and documentation *IEEE Trans. on Software Engineering*, vol.28, no.10, pp.970–983, 2002.
[18] A. Marcus, J. I. Maletic, Recovering documentation-to-source code traceability links using latent semantic indexing, *Proc. of 25th Inte'l Conf. on Software Engineering*, Leipzig, Germany, pp.125–135, 2008.
[19] L. R. Biggers, C. Bocovich, R. Capshaw, B. P. Eddy, L. H. Etzkorn, and N. A. Kraft, Configuring latent dirichlet allocation based feature location, *Empirical Software Engineering*, vol.19, no.3, pp.465–500, 2014.
[20] R. Oliveto, M. Gethers, D. Poshyvanyk, A. De Lucia, On the equivalence of information retrieval methods for automated traceability link recover, *Proc. of 18th Int'l Conf. on Program Comprehension*, Minho, Portugal, pp.68–71, 2010.
[21] Z. P. Fry, D. Shepherd, E. Hill, L. Pollock, and K. Vijay-Shanker, Analysing source code: looking for useful verbdirect object pairs in all the right places, *IET software*, vol.2, no.1, pp.27–36, 2008.
[22] D. Shepherd, L. Pollock, and K. Vijay-Shanker, Towards supporting on-demand virtual remodularization using program graphs, *Proc. of 5th Int'l Conf. on Aspect-oriented Software Development*, New York, USA, pp..3–14, 2006.
[23] A. S. Andrian Marcus, J. I. Maletic, Recovery of traceability links between software documentation and source code, *International Journal of Software Engineering and Knowledge Engineering*, vol.15, no.5, pp.811–836, 2005.
[24] S. C. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. A. Harshman, Indexing by Latent Semantic Analysis, *Journal of The American Society for Information Science and Technology*, vol.41, no.6, pp.391–407, 1990.
[25] J. Wang, X. Peng, Z. Xing, and W. Zhao, Improving feature location practice with multi-faceted interactive exploration, *Proc. of 35th Int'l Conf. on Software Engineering*, San Francisco, USA, pp.762–771, 2013.