Front.Comput.Sci.
DOI

RESEARCH ARTICLE

Correlation-based software search by leveraging software term database

Zhixing Li, Gang Yin (✉), Tao Wang, Yang Zhang, Yue Yu, Huaimin Wang

National Laboratory for Parallel and Distributed Processing, College of Computer, National University of Defense Technology, Changsha 410073, China

Abstract Internet-scale open source software (OSS) production in various communities generates abundant reusable resources for software developers. However, finding the desired and mature software with keyword queries from a considerable number of candidates, especially for the fresher, is a significant challenge because current search services often fail to understand the semantics of user queries. In this paper, we construct a software term database (STDB) by analyzing tagging data in Stack Overflow and propose a correlation-based software search (CBSS) approach that performs correlation retrieval based on the term relevance obtained from STDB. In addition, we design a novel ranking method to optimize the initial retrieval result. We explore four research questions in four experiments, respectively, to evaluate the effectiveness of the STDB and investigate the performance of the CBSS. The experiment results show that the proposed CBSS can effectively respond to keyword-based software searches and significantly outperforms other existing search services at finding mature software.

Keywords software retrieval; software term database; open source software.

1 Introduction

Software reuse is essential to improving the quality and efficiency of software development [1]. With the rapid development of the open source movement, substantial amounts of open source software (OSS) have been published through the internet [2]. For example, SourceForge[1)] has more than 460 thousand projects, GitHub[2)] has more than 30 million repositories, and the number of projects in these communities increases dramatically every day. On the one hand, this growth provides abundant reusable resources [3, 4]; on the other hand, the growth introduces a significant challenge for locating desired projects among numerous candidates.

Many project hosting sites, such as SourceForge and GitHub, have provided OSS search service to help developers identify candidate projects. General search engines, such as Google and Bing, are alternative choices because of their powerful query processing ability. However, the preceding search approaches are unsuitable for search scenarios where developers use short keyword queries to express their need for a software project for specific functionality and application context. For example, users may query "java ide" when they need to write Java programs, or query "python orm" when they program with python and prefer ORM engines to SQL statements. The preceding example is a common phenomenon found in developers who have insufficient development experience or those who possess a few programming skills but are new to the domain.

Search technologies applied by project hosting sites and general-purpose search engines are mainly based on text matching and are augmented with additional information, such as the hyperlink structure of the Web [5–7]. However, the problem is that some software project information may not explicitly appear in the original description. Furthermore, a few terms in these queries are too general, leading to the

Received month dd, yyyy; accepted month dd, yyyy

E-mail: yingang@nudt.edu.cn

[1)] sourceforge.net
[2)] github.com

matching of irrelevant information. Project hosting sites and general-purpose search engines cannot semantically and effectively understand user queries. Moreover, general-purpose search engines usually return related hyperlinks and insufficiently support direct answers; hence, users have to extract and conclude valuable information by themselves.

Precisely understanding the intention of users based on search keywords is a significant task. To solve the aforementioned problem, we introduce the correlation-based software search (CBSS), a correlation based method, to improve keyword-based software searches. Particularly, we construct a software term database (STDB) by leveraging tag-related data on Stack Overflow and respond to submitted queries with a correlation search based on the STDB.

A few construction approaches for software specific word database have been presented in recent years. For example, Howard et al. [8,9] mined semantically similar words in code and comments. Wang et al. [10] inferred semantically related terms based on their proposed similarity metric for software tags in FreeCode. Tian et al. [11] characterized each word using a co-occurrence vector that captures the co-occurrence of this word with the other words, all extracted from posts in Stack Overflow, to measure the similarity of two words.

In this paper, we mined term relevance from two perspectives. We first analyzed the co-occurrence of tags. By co-occurrence, we refer to two tags that label the same question. We utilized the co-occurrence frequency of tags and obtained their explicit relevance. We also utilized duplicate posts tagged by two different tag lists and obtained the additional implicit relevance of tags with these duplicate post links. We explored four research questions in four experiments, respectively, to evaluate the effectiveness of the STDB and investigate the performance of the CBSS.

RQ1 How well does STDB measure the relevance among software terms?

RQ2 Does CBSS improve the search for more relevant software projects compared with existing search services?

RQ3 Do the software projects returned by CBSS have a high degree of maturity?

RQ4 What is the actual effect of the SNS algorithm on the search results?

These experiments are conducted on 46,279 tags, 32,209,817 posts, and 3,089,262 post links contained in the Stack Overflow data dump released in September 2016 and more than 17 million projects hosted on GitHub and Oschina.

In summary, the following are the main contributions of the present paper.

1) We effectively analyze term relevance from different perspectives by leveraging tagging data on Stack Overflow and build a software term database.

2) We propose the CBSS, a novel technique based on the STDB, to improve the performance of keyword-based software searches.

3) We design a local re-rank method through similar and neighbor synergy (SNS) to improve the initial retrieval result.

4) We explore the performance of the CBSS using a comparative method and a user study, which illustrate that the CBSS can benefit software development by helping users to efficiently find mature software.

The rest of the paper is organized as follows. Section 2 introduces the preliminaries of our study. Section 3 elaborates on the CBSS. Section 4 provides details regarding the experimental setup. Section 5 presents the experimental results and analysis. Section 6 states the threats to validity. Section 7 shows the related works. Section 8 concludes this paper and introduces future work.

2 Keyword-based Software Search and its challenges

2.1 Definition of keyword-based software search

First, distinguishing between two very different kinds of software searches is necessary.

- *Target searches*: In this class of searches, users provide search engines with the name of a few software projects and expect to obtain its description in detail. However, this kind of search is not our main focus.
- *Seeking searches*: In many other scenarios, users need to find a few specific software projects by offering their need to search engines. This is the kind of searches we are interested in.

Moreover, web search engine users prefer to express information needs using short keyword queries [12, 13]. When users carry out seeking searches, these queries are usually generated by users to describe what application scenario a software project is about for application or what functional feature a software project should provide.

With the rapid development of OSS, numerous software projects of the same functionality have been produced. Many ways to evaluate a software project exist; these methods utilize source codes, documents, and other data in the software development process [14–17]. However, we believe that community feedback regarding a software project provides a more effective evaluation [18, 19]. Hence, we define

Fig. 1 Top search result for "java message queue" returned by OpenHub

the mature software as projects that remain in current and widespread use and received extensive discussion.

Therefore, the seeking search in keyword form for mature software is the focus of this study.

2.2 The challenges of keyword-based search in OSS

A few existing practical keyword-based software search cases are performed on the current service. Suppose that a developer is required to build a distributed web crawler with Java. The developer tends to use a message queue to dispatch the URLs of the web page to be crawled to achieve this. Then the query "java message queue" will be searched. Figure 1 is the top result of what the developer obtained from OpenHub. From the figure, we find that the most common Java message queues, RabbitMQ and ActiveMQ, are not shown in the results. We can see that current software project hosting platforms hardly return any popular and common software project relevant to the query. As for the search result returned by general search engines, a few items linked to the home page of the relevant software exist. However, the items are mixed with links that are irrelevant to any target software or that are relevant to a few projects but cost more time to analyze. We find that regular search engines return search results containing scrambled information.

We conclude based on the previous analysis that the following challenges of the keyword-based software search for current search services exist:

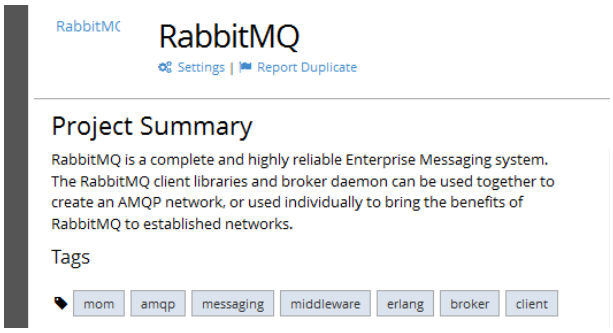


Fig. 2 Description of RabbitMQ in OpenHub

- A few terms in these queries are too general, thus matching a large number of irrelevant information.
- Application information or functionality description of a few software projects may be lost in its metadata. For example, Figure 2 shows the description of RabbitMQ in OpenHub, from which we cannot find any word regarding *"message queue"* that RabbitMQ usually acts as or regarding *"Java"* which belongs to client programming languages that RabbitMQ supports. Thus, it badly performs if a system only matches the words typed by the user with the text contained in software metadata.
- Criteria used to rank retrieval result are insufficient. The text match shows that project hosting sites usually use statistical data obtained from their own sites to rank the retrieval result, whereas general search engines analyze the link relation among web pages rather than software projects. Their ranking strategies inadequately reflect the real quality and usage status of software projects.

3 MEHTODOLOGY

We leverage software engineering social content and construct a software term database to support the keyword-based software search and solve the challenges described in Section 2. The overall process of the CBSS is illustrated in Figure 3.

First, we build a software term database and collect software metadata from the Internet. Then, provided with an initial query, three main steps should be done: query normalization, including query preprocessing and synonymy substitution; a correlation search comprising term expansion and software retrieval; and local re-ranking. Finally, a list of target software that best addresses the need of the developer will be produced. In the following subsections, we will explain the stages of the CBSS in detail.

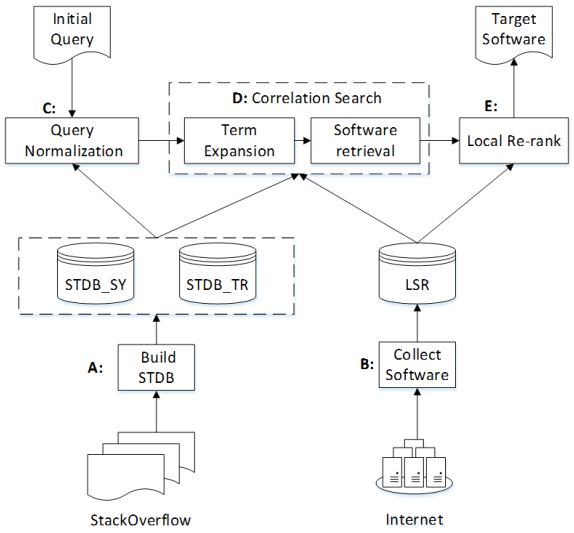


Fig. 3 The overall process of CBSS

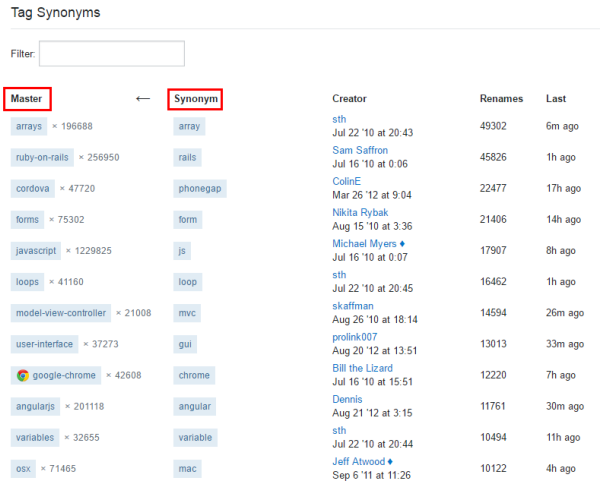


Fig. 4 Tag synonyms on Stack Overflow

Fig. 5 An example post in Stack Overflow

3.1 Software term database building

We construct the STDB based on the tags in Stack Overflow. We study Stack Overflow because it is a popular Q&A community that focuses on computer technology and is used by more than a million developers to discuss computer programming [20]. The tags in Stack Overflow are of high quality and are strongly related to software engineering, making them the best choice to build a term database of software engineering.

The STDB comprises two parts: STDB_SY and STDB_TR. STDB_SY contains common synonyms in software development experience, whereas STDB_TR includes term pairs that have explicit or implicit relevance in our settings.

3.1.1 Software term database of synonyms

When users want to issue a question, they are required to specify at least one tag that is representative of the question or that extensively describes the domain to which their question belongs. A tag is a keyword or label that categorizes questions; therefore, using the correct tags improves the efficiency of the search. However, developers usually prefer to use abbreviations to express a few specialized vocabularies, such as *"db"* for *"database"* and *"js"* for *"javascript"*. This convenience also results in a serious problem in the query process and usually leads to a vocabulary mismatch [21]. Therefore, we plan to build a software term database of synonyms and convert abbreviations, such as *"db"* and *"js"*, to their corresponding master forms, *"database"* and *"javascript"* respectively.

Tags in Stack Overflow are built by the community, which is commonly known as a folksonomy. They conducted some pre-seeding with a few dozen tags that are very obvious and clear, and most of the present tags were created by users in a collaborative process. Notably, not everyone has the right to create a new tag if he or she does not reach the threshold of reputations, which has slowly increased in Stack Overflow. In addition, they also have a community-driven tag synonym system, as shown in Figure 4. Hence, STDB_SY is dumped from Stack Overflow rather than WordNet [22]. This accurately reflects what synonym developers are using in their daily development. Currently, 2461 synonyms are contained in STDB_SY.

3.1.2 Software term database of term relevance

3.1.2.1 Explicit relevance

We first use tag co-occurrence to construct STDB_TR. The concept of tag co-occurrence is defined based on the common posts that two tags have tagged in Stack Overflow. Figure 5 shows an example of a post concerning the reliability of applying Redis instead of RabbitMQ. As illustrated, this post contains two tags: *"redis"* and *"message-queue"*. The two tags co-occur because they appear together in the tag list of this post.

Users are suggested to specify tags that are representative of the question or extensively describe the domain to which their question belongs [23, 24]. Hence, the name of a software project tends to co-occur with tags that correspond to its features or applications scenario. Thus, tag co-occurrence provides valuable term-relevance information.

We scanned every item in table posts and parsed the tags if the post was a question . Let $t_1, t_2, t_3, \ldots, t_k$ be the tag list of a specific question. We believe that every two tags paired in this list co-occur once. After the scan is conducted, the total incidences of co-occurrence of every two tag pairs will be counted. In total, we collected 3,719,128 word pairs.

Finally, triples in a form, such as $< t_1, t_2, values >$, will be stored into STDB_TR. In the triple, t_1, t_2 indicate two tags, and value is their relevance value. Given two tags, t_1 and t_2, their relevance value is computed by the following formula:

$$Rv_{CO}(t_1, t_2) = \frac{co - occure(t_1, t_2)^2}{count_{pst}(t_1) * count_{pst}(t_2)} \quad (1)$$

In the preceding formula, t_1, t_2 indicate two tags, and $Rv_{CO}(t_1, t_2)$ is their relevance value in terms of co-occurrence. Function $count_{pst}(t)$ returns the number of questions tagged by tag t, and $co-occure(t_1, t_2)$ computes the number of questions tagged by t_1 and t_2. The numerator is the square of the number of questions that two tags tagged together. The denominator is the product of the numbers of questions that the query item and mutual term tags, respectively. $Rv_{CO}(t_1, t_2)$ will be 1 if two tags always tag the same question, and 0 if they tag totally different questions.

3.1.2.2 Implicit relevance

STDB_TR is also enriched by analyzing duplicate posts on Stack Overflow. Although Stack Overflow encourages users to "search and research" the existing knowledge base before they post a new question, the difficulty in finding the necessary information may still result in duplicate questions. Such questions are explicitly marked as duplicates by users with a high reputation and can be easily recognized by the [duplicate] marker at the end of the question title. Figure 6 provides an example of duplicate posts. The preceding question (ID = 2229329) is marked as a duplicate of the question below (ID = 139639). Both questions discuss the difference between two SQL commands, "TRUNCATE" and "DELETE".

The post that is marked as a duplicate is called a source post, and the post which a duplicate post is linked to is called a target post. Clearly, the two linked posts talk about the same thing. Hence, the tags of a source post and the tags of

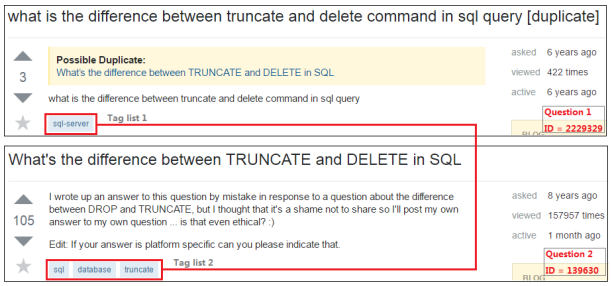


Fig. 6 An example duplicate post in Stack Overflow

a corresponding target post of this source post are also relevant. This relevance underlies the implicit relevance . Given a source post ps and the corresponding linked target post pt, two tag lists are present: $ts < ts_1, ts_2, \ldots, ts_m >$ for ps, and $tp < tp_1, tp_2, \ldots, tp_n >$ for pt (m and n indicate ps has m tags and pt has n tags, respectively). We first compute the common tags set tm in ts and tp: $tm = ts \cap tp$. With tm, we can obtain $ts\_m = ts - tm$ and $tp\_m = tp - tm$. For each tag in ts_m, we built an undirected relevance link to each tag in tp_m to make the best of information that is contained in ts_m and tp_m. The number of times each tag pair has been linked will be counted after all duplicate posts have been processed. In conclusion, 131,919 tag pairs have implicit relevance.

Indeed, implicit relevance can be regarded as a complement to explicit relevance, and the linked times of each two tags can be counted into the co-occurrence of the two tags. Hence, we prefer to update STDB_TR as follows:

$$Rv(t_1, t_2) = \frac{(co - occure(t_1, t_2) + count_{dpl}(t_1, t_2))^2}{count_{pst}(t_1) * count_{pst}(t_2)} \quad (2)$$

Function $count_{dpl}(t_1, t_2)$ returns the number of times $t_1 and t_2$ have been linked due to duplicate posts.

3.2 Software collection

We collect software projects from two sources, including Oschina and GitHub. The projects hosted in Oschina are collected by a web crawler that we developed, which is illustrated in Figure 7. Oschina shows all the included projects in the form of a list. Therefore, our crawler first crawled list pages and downloaded the detail pages whose link is extracted from the crawled list pages.

We collect projects hosted on GitHub by downloading a dump of GHtorrent[3)] released on June 1, 2016, which provides an offline mirror of the repositories of GitHub. From

[3)] www.ghtorrent.org

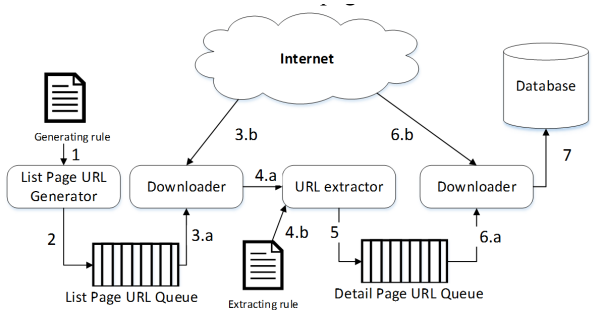


Fig. 7 The process of our crawler for Oschina

all the repositories, we select the original projects that are not forked from the other projects.

The preceding subsections concern offline work. In the following steps, we focus on what happens to a given query.

3.3 Query normalization

In this step, we first apply some typical text preprocessing, such as tokenize and stop words removal, on the initial query. Then we convert all the words to their lower-case and primary form, for example, *"JavaScript"* to *"javascript"* and *"databases"* to *"database"*. Finally, we combine neighbor items in the initial query to attempt more possibility for matching the software term database to deal with the writing style of the user. For example, *"java message queue"* can also be expressed as *"java-message queue"*, *"java message-queue"* and *"java-message-queue"*. After preprocessing, we normalize the items found in the database to their corresponding master form. STDB_SY is used to achieve this goal, and every synonym is replaced by its master form if it has one.

3.4 Correlation search

3.4.1 Term expansion

After normalization, a query can be donated as $q = t_1, t_2, t_3, \ldots, t_k$ where $t_i (1 \leq i \leq k)$ is a valid term after the preceding process and k is the number of valid term of query q. The CBSS first scans STDB_TR and fetches the terms related to each term in q to perform the correlation search. The elated terms of t_i can be denoted as $Rt(t_i)$. We assume that terms appearing in each related tag list of query terms can possibly be the real intent for the user who typed that query. We call such terms as mutual terms. The following formula returns the mutual terms expanded for query q.

$$Mt(q) = \bigcap_{t_i \in q} Rt(t_i) \tag{3}$$

3.4.2 Software retrieval

Each expanded mutual term will be retrieved in LSR with textual matching between the term and the name of the software. This retrieval process produces the initial unsorted result. We proposed a ranking model to rank the unsorted result, that is, determine which one is more likely to satisfy the need of the user. Every candidate software is assigned a rank score under this model, which is computed by the following formula:

$$Rc(cs) = \prod_{i=1} Rv(cs, t_i) \tag{4}$$

In addition, retrieved projects should be reusable software frameworks or development tools rather than programming languages, domain terminologies, and concepts that are extensively considered as software project but are too fundamental to appear in the search result. Meanwhile, we have classified and concluded three categories of such terms: Operating System, Programming Language, and Software Standard, which are shown in Table 1. The Operating System category contains computer operating systems (e.g., Linux, Windows) and mobile operating systems (e.g., Android, iOS). The Programming Language category covers different types of known programming languages, such as object-oriented (e.g., Java), scripting (e.g., Python), and markup (e.g., Python)). The Software Standard category refers to data formats (e.g., JSON), protocols (e.g., HTTP), and so on.

Table 1 Filter categories and examples

categories	examples
operating system	e.g., Linux, Android, Window, iOS
programming language	e.g., Python, Java, Ruby, XML
software standard	e.g., JSon, OAuth, HTTP, Java-EE

.

3.5 Local re-rank

After the refinement process, the local re-rank process is performed to improve the initial retrieval result. The underlying idea of our local re-rank algorithm is as follows: if two similar software projects rank very differently, that is, the distance between their positions in the initial retrieval result list is relatively long, one of them is out of position. Hence, adjusting them to their proper positions is necessary. The following

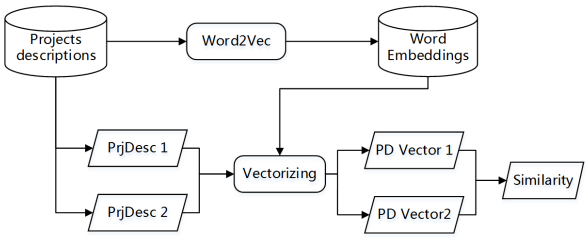


Fig. 8 The process to compute similarity between two projects

steps are necessary to achieve this: measuring project similarity and re-ranking the initial trivial result with our SNS algorithm.

3.5.1 Project similarity measuring model

We apply natural language processing (NLP) and neural network techniques to measure the similarity of two software projects. Figure 8 shows that we treat a software project and its description in Stack Overflow, which comprise the excerpt and wiki of a tag, as a document denoted by 8. Therefore, the problem is converted to determine the similarity between the two documents.

First , we use a continuous skip-gram model [25], which is one of the state-of-the-art neural network techniques for capturing word-level semantics for NLP tasks, to learn domain-specific word embeddings from the text of *PrjDescs* collection. The output is a dictionary of word embeddings for each unique word.

Word embeddings encode semantics at word-level. Capturing semantics at the document level is necessary to compute *PrjDesc* semantic relatedness. Given two *PrjDescs*, they are first tokenized and converted into vectors of the word. Meanwhile, each word in the vectors is represented as a vector by searching the word embeddings dictionary, which is then added together and averaged to produce a PD vector for each *PrjDesc*. Finally, the similarity of the two *PrjDesc* is computed by the cosine similarity of their *PD vectors*.

3.5.2 Similar and Neighbor Synergy algorithm

In this paper, we introduce the SNS algorithm to re-rank the initial retrieval *P*. As Algorithm 1 shows, the SNS algorithm comprises five main steps.

Line 1: We first produce *prjPs* which is a collections of projects pairs, such as *(prj_m, prj_n)*, wherein *prj_m∈P, prj_n∈P, 0<=m<len(P) and m<n<len(P)*.

Line 2-5: We find candidate project pairs in *prjPs* with two thresholds: *threshold_sim* and *threshold_pos*, which are set to 0.85 and 3, respectively, in our experiment. Herein is a mathematical relationship between *threshold_pos* and *threshold_sim*:$threshold\_pos = \lfloor (1 - threshold\_sim) * t/(1 - b) \rfloor$, where t and b are observation values. The parameters t and b mean that the difference of rank position of two software whose similarity is b should be less than t. We set the parameter t to 10 because we study the top 10 results. After the observation of our gold sets (GSs), we determine that b is 0.55. Candidate project pairs are determined if their rank distance is larger than *threshold_pos*, and the similarity computed by M, which measures the similarity of two projects, is larger than *threshold_sim* . In one item of the candidate project pairs, one of the two projects is out of position if its neighbor similarity is larger than the other one's, Neighbor similarity of a project is the average value of pair wise similarity between *prj* and each of its neighbors. In addition, if *prj_i* is out of position, then the effect of *prj_j* is to pull down the rank of *prj_i*. On the other hand, if *prj_j* is out of position, then the effect of *prj_i* is to push up the rank of *prj_j*. At the end of this step, every out-of-position project together with its affected projects will be added to a set named *oopPrjs*. Simultaneously, all the remaining projects that are not needed to adjust rank position will be put to the set *stayPrjs*.

Algorithm 1 Similar and Neighbor Synergy algorithm for local re-rank

Input: *P* is the initial retrieval result
Output: the final ranking result
1: *prjPs* = producePrjPs(P)
2: for *(prj_i, prj_j) ∈prjPs* do
3: check if *prj_i (prj_j)* is out of position;
4: put *prj_i(prj_j)* into *oopPrjs* or *stayRanks*;
5: end for
6: for *prj ∈oopPrjs* do
7: calculate new rank for *prj*;
8: end for
9: merge *stayRanks* and *oopPrjs*
10: **return** the merged result

Line 6-8: All the affecting projects of an out-of-position project will be sorted by the original rank in *P*, and induce an iterative effect on this project. Whether through push-up or pull-down effect, the affecting project will take the out-of-position project closer by half of the current rank distance. After completing all actions, the new rank of the out-of-position project is determined.

Line 9: Projects in *stayPrjs* or *oopPrjs* all correspond to a rank position. The final step is to merge the projects into

the final rank list. However, a problem occurs when merging these two sets: the desired rank position of a project in *oopPrjs* may conflict with the desired rank position of the other one in *stayPrjs* or *oopPrjs*. Our strategy is that, if the conflicted projects are all in *oopPrjs*, they will be ranked by original rank position in P, otherwise projects in *oopPrjs* is of higher priority than those in *stayPrjs* and success to take the conflicted rank position first.

Line 10: Finally, the merged list of target projects will be returned as the final rank list.

4 EXPERIMENTAL SETUP

In this section, we evaluate the effectiveness of the STDB and CBSS for improving the performance of keyword-based software retrieval based on four research questions (RQs). We conduct four experiments to answer the four questions. In addition, we introduce the dataset employed in the experiments.

4.1 Research Questions

We have the following research questions:

RQ1: How well does STDB measure the relevance among software terms?

First, we want to inspect the effectiveness of the STDB. We conduct a user study to answer this question. Users freely determine software terms that will be used for the retrieval of relevant terms from the STDB. Then they will be asked to evaluate whether the relevance values correspond to the truth.

RQ2 Does CBSS improve the search for more relevant software projects compared with existing search services?

Our second experiment compares the CBSS and general-purpose search engines and project hosting sites in terms of result relevance. We want to identify based on the results whether the CBSS can indeed find more relevant software projects. More specifically, selected general purpose search engines includes Google, Bing and Baidu which have been widely used in people's daily life and SourForge, OpenHub, GitHub, and Oschina, which are representative project hosting sites, are chosen in the experiment.

In addition, we collect a few typical query scenarios in the form of keyword searches for software projects from volunteers and use them to assess the CBSS and other search services. Table 2 shows all these search scenarios, which cover three categories: development tools, reusable libraries, and operational tools.

Table 2 Search scenarios

ID	key word	category
1	Java ide	Development tool
2	Dependency management	
3	Nosql database	
4	Java message queue	
5	Java logging	Reusable libraries
6	Python machine learning	
7	Game engine	
8	Js visualization	
9	Ruby orm	
10	Web spider	
11	Deep learning	
12	CI	Operational tool
13	Docker cluseter	
14	Linux monitoring	
15	http server	

.

RQ3 Do the software projects returned by CBSS have a high degree of maturity?

We conduct the third experiment to further study the maturity of the retrieval result. The software with high usage and attention tend to gain high usage because people are liable to choose extensively tested and applied software. We call this software as mature software. The ability to return more mature software rather than immature software is an important quality for search services.

RQ4: What's the actual effect of the SNS algorithm on the search results?

Finally, we want to examine how the SNS algorithm affects the search result. The experiment is conducted to compare the difference between the results when the algorithm is applied and the results when the algorithm is not applied in terms of the relevance and maturity of retrieved software.

4.2 Dataset collection

In this subsection, we describe the dataset in which SDTB and LSR are built.

We construct the STDB based on one version of Stack Overflow data dumps, which contains the data produced from July 31, 2008 to September 18, 2016. We also collect software projects from Oschina and GitHub to LSR.

The data statistics are shown in Table 3. After scanning table tags we obtain 46, 279 tags. We parse two tables, posts and post_links, to build STDB_TR. There are 32,209,817 posts and 3,089,262 links in table posts and post_links re-

spectively. Each question or answer is referred to as a post in Stack Overflow. We also use the APIs provided by the Stack Exchange to build STDB_SY. We have collected more than 17 million software projects.

Table 3 Data source

dataset	num. of items
tags	46, 279
posts	32, 209, 817
post links	3, 089, 262
software projects	17, 276, 862

.

4.3 Evaluation metrics

To answer the preceding research questions, our evaluation comprises three aspects: accuracy, relevance, and usability.

4.3.1 Accuracy evaluation

In the user study, volunteers are asked to evaluate how well the STDB reflects software term relevance. The evaluation is a Likert-type scale with a more detailed expression for each choice [26]. The respondents are asked to choose one of three candidate response items, that is, our evaluation process is a three-point Likert scale. Table 4 describes these three candidates.

Fifteen individuals with different backgrounds were invited to evaluate the results. Among them, seven are master students, three are Ph.D. students, and five are engineers with at least three years software development experience.

Table 4 Likert scale response categories for user evaluation

scale	response category
3	perfectly reflect term relevance
2	partly reflect term relevance
1	hardly reflect term relevance

.

4.3.2 Relevance evaluation

Numerous measures are available to assess the performance of a search method. One of these measures is MAP [27], which is a common measure in ranked retrieval results. For a single query, average precision (AP) is the average of the precision value obtained for the top k documents in retrieval results. This value is then averaged over queries to obtain MAP. MAP is computed by the following formula:

$$MAP(Q) = \frac{1}{|Q|}\sum_{j=1}^{|Q|}\frac{1}{m}\sum_{k=1}^{m_j} Precision(R_{jk}) \quad (5)$$

In the formula, R_{jk} is the set of items in the retrieved results from the top until the K_{th} relevant item is reached, whereas m indicates the total number of relevant items for queryQ_j. For a given query, returning the relevant software and ranking the relevant software in the top of the retrieval result are important. In a web search, as stated by studies [27, 28], people tend to prefer results located on the first page, or at most, the first three pages. This leads to another measure precision at k items (also known as P@k) that measures the precision at a fixed number of retrieved results, such as 10 or 20 items. Therefore, we combine AP and P@k and use measure AP@k which is similar to AP but is located at a fixed level of the retrieval result rather than recall level. In our case, we adopt the advice from the study [27] and set k to 10. That is, we will compare the top 10 results returned by each approach.

4.3.3 Usability evaluation

When we collect queries from developers, we also ask them to recommend target software projects which will be put into corresponding gold set after manual validation. In addition to this, we also search on the Internet and look for eligible software through various information platforms like BBS, blogging and Q&A websites. Based on all the candidate software projects, we construct the final gold set by considering the actual usage and community reputation and retaining the top ten eligible software. And then these gold sets will be used to measure Recall and AP for every search service. Recall reflects how many relevant and mature software projects are returned in the retrieval result.

5 RESULTS AND ANALYSIS

In this section, we discuss and analyze four experiment results and their evaluations with regard to the RQs proposed in Section 4.1.

5.1 RQ1: How well does STDB measure the relevance among software terms?

In our user study, we collect 82 query terms from 15 participants. Table 5 shows the Likert score distribution of this experiment. The average score is 2.81, which means that the STDB can effectively reflect relevance among software terms.

Table 5 Statistics of user study result on STDB

score	number of choice
3 (excellent)	69
2 (moderate)	11
1 (poor)	2
average score:	2.81

.

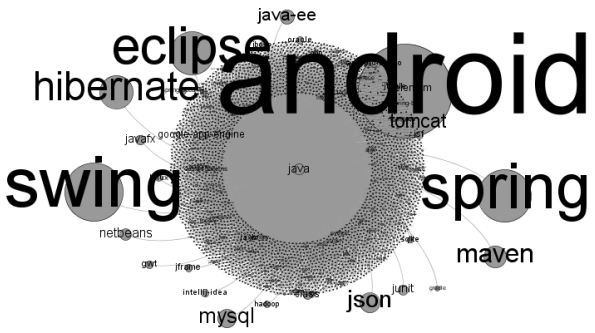


Fig. 9 Relevant terms of "java"

Figure 9 shows an example and presents the relevant terms for "java". The central node indicates that the term "java" and the other nodes that are connected to it represent terms relevant to "java". The size of the node corresponds to the relevance values between the node and the "java" node. The bigger the node, the more relevant a node is to the "java" node.

Result 1: *STDB gets a score of 2.81 out of 3 in the user study which means STDB can indeed effectively measure the relevance among software terms*

5.2 RQ2 Does CBSS improve the search for more relevant software projects ?

As previously mentioned, two kinds of comparisons are conducted in the first experiment to answer RQ2: (1) CBSS vs. software project hosting sites, and (2) CBSS vs. general-purpose search engines. Table 6 shows the comparison result of AP@10 for each search scenario. Each row in Table 6 shows the AP@10 of each search scenario for every search service, and the cell in which the number is in bold indicates the service that performs best for this scenario. Take query #6, which is *"python machine learning"*, as an example: 7 out of 10 retrieval results of the CBSS are relevant, and the corresponding AP@10 is 0.69, while other approaches find relatively few relevant software projects which results in low performance.

Table 6 AP@10 for each search systems

ID	CS	OS	OH	SF	GH	GL	BN	BD
1	**0.50**	0.15	0.38	0.07	0.00	0.21	0.24	0.10
2	0.51	0.10	0.26	0.03	0.76	0.12	0.12	0.12
3	**0.69**	0.27	0.39	0.04	0.20	0.14	0.14	0.04
4	**0.73**	0.46	0.17	0.24	0.54	0.20	0.14	0.00
5	0.24	**0.90**	0.20	0.00	0.26	0.05	0.02	0.00
6	**0.69**	0.00	0.27	0.08	0.10	0.05	0.00	0.10
7	0.89	0.76	0.78	0.41	**1.00**	0.14	0.05	0.05
8	**0.43**	0.05	0.26	0.03	0.24	0.50	0.27	0.04
9	0.30	0.47	0.11	0.00	**0.52**	0.30	0.05	0.00
10	0.64	0.20	0.34	0.57	**0.68**	0.17	0.07	0.05
11	**0.90**	0.17	0.55	0.05	0.27	0.00	0.00	0.00
12	**0.57**	0.20	0.31	0.00	0.30	0.01	0.00	0.10
13	0.18	0.20	**0.54**	0.00	0.09	0.15	0.05	0.00
14	0.26	0.29	0.29	0.19	**0.64**	0.11	0.10	0.00
15	0.44	0.53	**0.88**	0.31	0.26	0.41	0.24	0.02

Abbreviations: CS: CBSS, OS: Oschina, OH: OpenHub, GH: GitHub, GL: Google, BN: Bing, BD: Baidu, SF: SourceForge .

5.2.1 CBSS vs. software project hosting sites

In the comparison between CBSS and project hosting sites, GitHub, SourceForge, OpenHub, and Oschina which are the most famous among these sites, are chosen.

A two-valued judgment system was used, and the two values are "not relevant" and "relevant". Every item in the retrieval results has been carefully scrutinized to determine whether it is relevant to the query. Software projects with a title or description that is relevant to the query but do not have any code are viewed as "not relevant".

Take the search scenario *"java message queue"* as an example. Table 7 presents the results of each method. The results show that the search service provided by project hosting sites usually retrieve by matching the textual query and text contained in the project title or description. This tends to result in superficial matching, wherein the textual query is contained or partly contained in the metadata of the retrieved project; however, the relevance between them is very low. In addition, most project hosting sites do not verify whether the returned projects are valid or the code is not empty. All these factors worsen the performance of these project hosting sites.

5.2.2 CBSS vs. general search engines

We select Google, Bing, and Baidu to perform the contrast analysis and compare the CBSS and general search engines. Results returned by general search engines are a list of URL

Table 7 Retrieval result for "java message queue"

search service	retrieval result
CS	***Activemq***, ***rabbitmq***, ***websphere-mq***, ***amqp***, ***apache-kafka***, *apache-camel, websphere,* ***hornet***, ***spring-amqp***, ***zeromq***
OS	***SUN Java System Message Queue***, ***HQueue***, ***Akka***, *Appserver.io,* ***mJMS***, ***Open Message Queue***, *Android package android-ActionQueue, Jetlang, NoHttp*
OH	*Java MQ Message Testing Tools, SAFMQ, New Java Fast Socket Message Server, simple-mq,* ***metis-jms***, *notify4j,* ***myqueue***, *ProMVC,Java_Examples, penSource .NET & Java Messaging Service*
SG	*Qmhandle, bacnet for java, mxa, weblogic mq, Java SMPP Client, jlibmodbus, gmail api for java, facebook auto group poster, beecrypt-cryptography-library,* ***activemq browser***, *java application framework for all*
GH	*aillard/mongo-queue-java, xetorthio/rmq, awslabs/amazon-sqs-java-messaging-lib, softlayer/softlayer-message-queue-java, scottbyrns/javascript-message-queue, azure samples/storage-queue-java-getting-started, li-robot/messagequeue,* ***jram13/message-queue***, *openstack-capstone/burrow-java, uyan/javamessagequeue*

Abbreviations: CS:CBSS, OS:Oschina, OH:OpenHub, GH:GitHub, . SF:SourceForge

of web pages, and we treat a URL as relevant if it links to the official or hosting site of a few relevant software. Interestingly, Google and Bing, which are world-renowned search engines, performed poorly. With regard to these general search engines, the vertical search for a software is a disadvantage, and they hardly return an official site of a particular software and instead provide many posts related to queries of users. Users may locate valuable information regarding related software after clicking and going through these posts; however, more clicks and more time are required. Filtering useful information from web pages by themselves is not convenient for users.

Finally, Figure 10 shows the MAP@10 of all these scenarios for each approach. In this measure, the CBSS performed the best and Baidu performed worst, hardly returning any relevant software projects. On average, the vertical search service provided by project hosting sites performs better than general search engines, whereas SourceForge lags

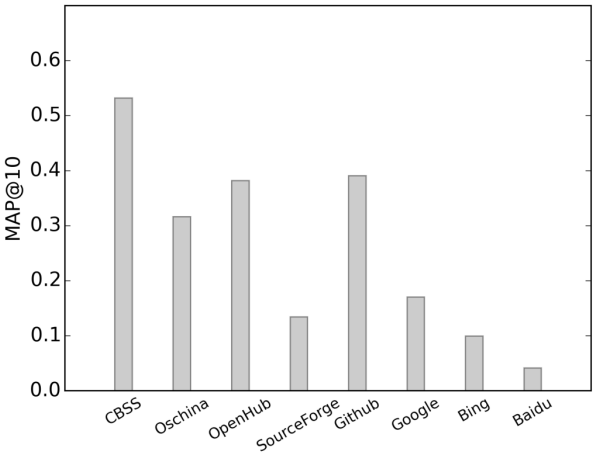


Fig. 10 MAP@10 of each search system

behind Google.

In the comparison between the CBSS and project hosting sites, the CBSS is inferior to others in a few search scenarios. For query #9 (*"ruby orm"*), the results returned by CBSS are shown in Table 8. The software project relevant to *"ruby orm"* is an ORM engine and can be simultaneously applied to the Ruby context.

Table 8 Top10 retrieval results for "ruby orm" returned by CBSS

rank	result	rank	result
1	ActiveRecord	6	Yaml
2	DataMapper	7	Validation
3	Sequel	8	MongoDB
4	PostgreSQL	9	SQLite
5	MySQL	10	Sinatra

.

Obviously, "ActiveRecord", "DataMapper" and "Sequel" are target projects; however, "PostgreSQL", "MySQL" and the others should not appear in the retrieval result. They have been returned because of their co-occurrence or their provision of links of duplicate posts with terms, "ruby" and "orm". On this point, they really should be in the result list. However, their relevance is distinct from the relevant software. The "greentDAO" is relevant to "orm" because it is a kind of ORM, while "MySQL" is relevant to "orm" because it is usually used together with an ORM project, resulting in a drift from the original search intention and decreasing the performance of the search. We consider this problem as a mismatch of intention. Although the local rank has improved this situation, more work can be conducted in the future.

Result 2: *For most search scenarios, the AP@10 of CBSS exceed other approaches and our MAP@10 is the most outstanding one which means CBSS is more likely to find users relevant software projects than other search approaches.*

5.3 RQ3 Do the software projects returned by CBSS have a high degree of maturity?

For each search system, we compute the average recall@top-k on all search results with k ranging from 1 to 10. As we can see from the experiment results shown in Table 9, our method always performs better than other search systems on returning mature software. Specifically, Table 10 illustrates the recall@top-10 on each search result. CBSS performs the best on all the search scenarios with only one exception on query #14.

Table 9 Average recall@Top-k of each search system

Top-K	CS	OS	OH	SF	GH	GL	BN	BD
1	**0.15**	0.05	0.05	0.02	0.01	0.06	0.01	0.03
2	**0.30**	0.06	0.05	0.02	0.02	0.08	0.09	0.04
3	**0.36**	0.10	0.10	0.03	0.03	0.14	0.13	0.04
4	**0.41**	0.10	0.14	0.04	0.04	0.17	0.13	0.04
5	**0.49**	0.10	0.17	0.06	0.06	0.18	0.17	0.05
6	**0.54**	0.15	0.17	0.08	0.06	0.19	0.18	0.06
7	**0.56**	0.21	0.17	0.09	0.07	0.20	0.19	0.06
8	**0.58**	0.22	0.19	0.10	0.08	0.21	0.19	0.06
9	**0.62**	0.23	0.21	0.12	0.08	0.23	0.19	0.06
10	**0.64**	0.23	0.21	0.13	0.08	0.27	0.19	0.07

Abbreviations: CS: CBSS, OS: Oschina, OH: OpenHub, GH: GitHub, . GL: Google, BN: Bing, BD: Baidu, SF: SourceForge

We also explore the change of MAP@10 when the experiment is conducted on the gold set rather than relevant set (RS).

When we say RS, we mean a project is included in RS as long as it is relevant to a query and it doesn't have to be a mature one. Actually, the second experiment and evaluation referenced this set. We compute MAP@10 on all the search results illustrated in Figure 11, which shows CBSS is still far ahead of the competition. In detail, Table 11 shows the volatility of MAP@10. CBSS gets relatively low decrease (35.71%) with the highest MAP@10 (0.56) on RS, which indicates CBSS tends to return more mature software projects in the search result.

Table 10 Recall@10 of each search system

ID	CS	OS	OH	SF	GH	GL	BN	BD
1	**0.80**	0.20	0.40	0.00	0.00	**0.80**	**0.80**	0.20
2	**0.33**	0.11	0.00	0.00	0.00	0.22	0.22	0.22
3	**0.71**	0.14	0.43	0.00	0.00	0.29	0.29	0.00
4	**0.80**	0.40	0.00	0.00	0.00	0.40	0.20	0.00
5	**0.60**	0.60	0.20	0.00	0.40	0.20	0.00	0.00
6	**0.88**	0.00	0.12	0.38	0.12	0.12	0.00	0.12
7	**0.43**	0.00	0.00	0.00	0.00	0.14	0.14	0.14
8	**0.60**	0.00	0.40	0.00	0.00	0.40	0.40	0.20
9	**1.00**	0.67	0.67	0.00	0.00	0.67	0.33	0.00
10	**0.60**	0.00	0.00	0.10	0.00	0.10	0.00	0.00
11	**0.60**	0.10	0.20	0.00	0.40	0.00	0.00	0.00
12	**0.60**	0.20	0.10	0.00	0.20	0.00	0.00	0.00
13	**0.57**	0.14	0.14	0.00	0.14	0.29	0.14	0.00
14	0.60	**0.80**	0.00	0.60	0.00	0.20	0.20	0.00
15	**0.86**	0.14	0.43	**0.86**	0.00	0.29	0.14	0.14

Abbreviations: CS: CBSS, OS: Oschina, OH: OpenHub, GH: GitHub, . GL: Google, BN: Bing, BD: Baidu, SF: SourceForge

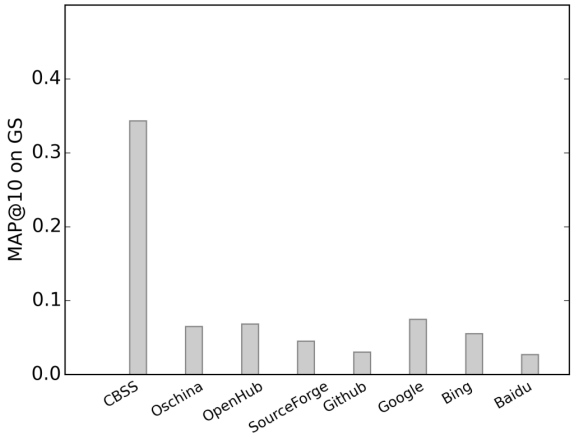


Fig. 11 MAP@10 of each search system

Result 3: *CBSS tends to return more software projects of high usability and so that are more likely to satisfy user's intent.*

5.4 RQ4: What's the effect of SNS algorithm?

We analyze the effect of SNS on the relevance and maturity of the search result. Figure 12 illustrates the change of the AP@10 of each search result after the SNS algorithm is applied. In terms of relevance, nine of the 15 search results improved, one remained unaffected, and five worsened, leading to an increase of 0.05 of AP@10 on average. With regard to maturity, six of the 15 search results improved, two remained

Table 11 The data volatility of MAP@10 on gold set

search system	MAP@10 on RS	MAP@10 on GS	Decrease of MAP@10
CBSS	0.53	0.34	35.84%
Oschina	0.32	0.07	78.12%
OpenHub	0.38	0.07	81.58%
SourceForge	0.13	0.05	61.54%
GitHub	0.39	0.03	92.31%
Google	0.17	0.07	58.82%
Bing	0.1	0.06	40.00%
Baidu	0.04	0.03	25.00%

.

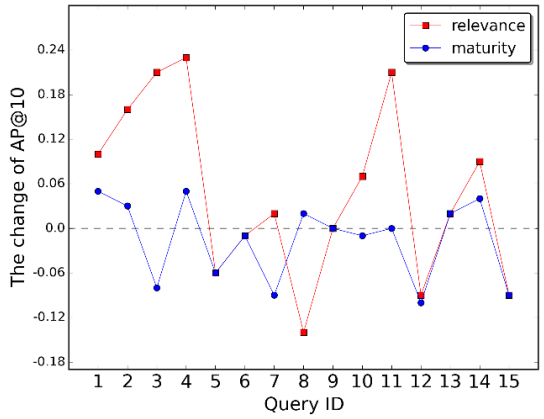


Fig. 12 The effect of SNS algorithm

unaffected, and seven worsened, resulting in a decrease of 0.01 of AP@10 on average. The results show that the SNS algorithm can improve the initial search result in terms of relevance with a slight negative effect on the usability of the CBSS. Therefore, SNS plays an active role in optimizing the initial search result.

Result 4: *The SNS algorithm has positive effect on the initial search result*

6 THREATS TO VALIDITY

This section discusses the threats to the validity of our work. They are listed as follows:

Search scenarios: We collect search scenarios as possible as we can to measure and compare the performance of the CBSS and other search methods. However, we may have failed to cover a few topics. In the future, we plan to collect search scenarios via crowdsourcing and the query logs of our upcoming platform, which analyzes and evaluates OSS on a worldwide scale.

Software project consistence: In the process of the CBSS, the association between the software development community and the knowledge sharing community depends on the name of the project. In most situations, this association is viable. However, in rare circumstances, this association may result in mistakes because of name ambiguity and differences in naming conventions. We have manually dealt with a few mistakes encountered during our experiments. Automatic software entity disambiguating is desired in future work.

Comparison fairness: When gathering search results from comparative search services provided by general-purpose search engines and project hosting sites, we try to limit the acquisition time of each site as thoroughly as possible for fair comparison. However, these systems, especially general-purpose search engines, always look for better service and update their algorithms to return more satisfactory results. Hence, an off-chance that the update of their algorithms may affect the search results exists.

User study & Gold set construction: As mentioned in Section 4.3, we recruit 15 participants to assess the maturity of software projects in the search result. The participants have several years of experience in program development. We believe that they can assign the correct scores with the assistance of their experience and the related information in the webpages. However, the participants are still at different programming levels and may assess the same project with different maturity scores. This threat can be minimized by group discussion and employing more experienced experts to arbitrate the assessment.

7 RELATED WORK

7.1 Software search

Software search is one of the basic tasks in software engineering [29]. Numerous papers have proposed various approaches to help users find software projects.

Tegawende et al. [30] proposed an integrated search engine that searches for project entities and provides a uniform interface with a declarative query language. Linstead et al. [31] developed Sourcerer to take advantage of the textual aspect of software, its structural aspects, as well as any relevant metadata.

Lu et al. [32] proposed an approach that expands a query with synonyms generated from WordNet and matches the expanded query with natural language phrases extracted from

source code identifiers. Nie et al. [33] reformulated the initial query by identifying software-specific expansion words in Stack Overflow to better solve the term mismatch problem. Lv et al. [34] applied the extended Boolean model to retrieve code snippets and considered both API understanding and text similarity matching. Collin et al. [35] proposed to consider the API calls and the data flow among those API calls in applications instead of only the descriptions of applications to help users find similar applications and examine how high-level concepts from queries implemented in the source code.

The preceding approaches mentioned either focused on software search in code level or restricted the form of query. Unlike these approaches, the CBSS in our study is focuses on the queries in the form of keywords for software in a higher level rather than just code snippets, such as ready-to-run software, third-party library, and so on.

7.2 Software-specific lexical database

Measuring the relation of words is essential to accurate software retrieval and recommendation. Sridhara et al. [36] conducted a comparative study and concluded that applying English-based semantic similarity techniques without any customization could be detrimental to software tools. Recently, a number of techniques and various measurements have been proposed to construct software-specific word databases.

Howard et al. [8] mined semantically similar words by mapping the main action verb from the leading comment of each method to the main action verb of its method signature. Yang et al. [9] inferred semantically related words in software by leveraging the context of words in the comments and code. The preceding work focused on text in source code file; however, many software-related words are not in the source code but are in the mass of content posted by developers and users.

Wang et al. [37] proposed a similarity metric to infer semantically related terms by considering their tagged documents tagged. In their metric, two kinds of similarity, that is, textual and document similarity, will be computed for each two software tags in FreeCode. The application of their database is limited because the scale is small. Tian et al. [11] measured the similarity of two words on the concept of word co-occurrence and constructed a word similarity database. They characterized each word using a co-occurrence vector that captures the co-occurrence of this word with other words all extracted from posts in Stack Overflow. Their last work [38] analyzed more text data from various sources. Like Tian's work, we also built a software term database with the data in Stack Overflow, but we chose high-quality data and that were specific to the software domain, that is, tags.

7.3 Social tags in Q&A sites

Numerous studies have analyzed the tags in Stack Overflow. Bhat et al. [39] studied the effect of tagging in response time for questions. They analyzed several factors and found that tag-related factors, such as the popularity of tags and subscribers, provide more valuable information than factors unrelated to tags. Studies [23, 24, 40] attempted to predict and recommend tags for the author of posts. They predicted tags by utilizing the content generated by users, historical tag assignments, and network properties in Stack Overflow. Mo et al. [41] inferred proper tags using the label propagation technique. They built semantic links between various URLs and tags to address partial tagging problems. Chen et al. [42] presented a new approach to recommend analogical libraries based on a knowledge base of analogical libraries mined from tags of millions of Stack Overflow questions.

8 CONCLUSION & FUTURE WORK

We first constructed the STDB based on different perspectives of the relationship between two software terms in Stack Overflow. We also proposed the CBSS, which performs a correlation search based on term relevance obtained in the STDB to improve the keyword-based software search. We also designed a novel local re-rank method to improve the initial retrieval result. We explored four research questions in four experiments, to evaluate the effectiveness of the STDB and investigate the performance of the CBSS. The evaluation of the empirical experiment shows that our CBSS significantly outperforms other search services. Compared with other search services, the CBSS can locate more mature software projects for users that are more likely to be helpful for the development of users.

The CBSS has been integrated to OSSEAN [2], a platform that evaluates OSS projects based on massive amounts of data across communities. Nevertheless, future work is necessary to improve the CBSS and enhance the ability of OSSEAN to search for software. First, a formal software domain knowledge graph that can provide more contribution to software engineering specific tasks should be established. Second, given that numerous other software domain terms can

be extracted from comments, post contents, and so on still exist, the text contents in Stack Overflow should be analyzed, and more terms and relationships should be extracted from them to further enrich the STDB.

Acknowledgements The research is supported by the National Natural Science Foundation of China (Grant No.61432020, 61303064, 61472430, 61502512) and National Grand R&D Plan (Grant No. 2016YFB1000805).

References

1. Frakes W B, Kang K. Software reuse research: Status and future. 2005, 31(7): 529–536
2. Yin G, Wang T, Wang H, Fan Q, Zhang Y, Yu Y, Yang C. Ossean: Mining crowd wisdom in open source communities. In: Service-oriented System Engineering. 2015, 367–371
3. Krueger C W. Software reuse. Acm Computing Surveys, 1992, 24(2): 131–183
4. Ghezzi C, Jazayeri M, Mandrioli D. Fundamentals of Software Engineering. China Electric Power Press, 2006
5. Haiduc S, Bavota G, Marcus A, Oliveto R, De Lucia A, Menzies T. Automatic query reformulations for text retrieval in software engineering. In: International Conference on Software Engineering. 2013, 842–851
6. Chau M, Chen H. Comparison of three vertical search spiders. Computer, 2003, 36(5): 56–62
7. Guha , McCool , Rob , Miller , Eric . Semantic search. Bulletin of the American Society for Information Science & Technology, 2003, 36(1): 700–709
8. Howard M J, Gupta S, Pollock L, Vijay-Shanker K. Automatically mining software-based, semantically-similar words from comment-code mappings. In: Working Conference on Mining Software Repositories. 2013, 377–386
9. Yang J, Tan L. Swordnet: Inferring semantically related words from software context. Empirical Software Engineering, 2014, 19(6): 161–170
10. Wang S, Lo D, Jiang L. Inferring semantically related software terms and their taxonomy by leveraging collaborative tagging. In: IEEE International Conference on Software Maintenance. 2012, 604–607
11. Tian Y, Lo D, Lawall J. Automated construction of a software-specific word similarity database. Automated construction of a software-specific word similarity database, 2014
12. Meij E, Balog K, Odijk D. Entity linking and retrieval for semantic search. In: ACM International Conference on Web Search and Data Mining. 2014, 683–684
13. Rasolofo Y, Savoy J. Term proximity scoring for keyword-based retrieval systems. Lecture Notes in Computer Science, 2002, 2633: 79–79
14. Widdows C, Duijnhouwer F. Open source maturity model. Cap Gemini Ernst & Young. New York NY, 2003
15. Wasserman A I, Pal M, Chan C. The business readiness rating: a framework for evaluating open source. EFOSS-Evaluation Framework for Open Source Software, 2006
16. BarbaraRusso , ErnestoDamiani , ScottHissam , BjÃűrnLundell , GiancarloSucci . Open Source Development, Communities and Quality. Springer US, 2008
17. Yu Y, Wang H, Yin G, Wang T. Reviewer recommendation for pull-requests in github. Information & Software Technology, 2016, 74(C): 204–218
18. Fan Q, Wang H, Yin G, Wang T. Ranking open source software based on crowd wisdom. In: IEEE International Conference on Software Engineering and Service Science. 2015, 966–972
19. Zhang Y, Yin G, Wang T, Yu Y, Wang H. Evaluating bug severity using crowd-based knowledge: An exploratory study. In: The Seventh Asia-Pacific Symposium on Internetware. 2015
20. Bhat V, Gokhale A, Jadhav R, Pudipeddi J, Akoglu L. Min(e)d your tags: Analysis of question response time in stackoverflow. In: Ieee/acm International Conference on Advances in Social Networks Analysis and Mining. 2014, 328–335
21. Pal D, Mitra M, Bhattacharya S. Exploring query categorisation for query expansion: A study. Computer Science, 2015
22. Miller G A. Wordnet: a lexical database for english. Communications of the Acm, 1995, 38(11): 39–41
23. Stanley C, Byrne M D. Predicting tags for stackoverflow posts. Proceedings of ICCM, 2013
24. Short L Z DW. C. Tag recommendations in stackoverflow. 2014
25. Mikolov T, Chen K, Corrado G, Dean J. Efficient estimation of word representations in vector space. Computer Science, 2013
26. Jamieson S. Likert scales: how to (ab)use them. Medical Education, 2004, 38(38): 1217–1218
27. Manning C D, Raghavan P, Tze H. Introduction to Information Retrieval. Posts & Telecom Press, 2010
28. Aula A, Majaranta P, Kari-Jouko . Eye-tracking reveals the personal styles for search result evaluation. Lecture Notes in Computer Science, 2005, 3585: 1058–1061
29. Hucka M, Graham M J. Software search is not a science, even among scientists. 2016
30. Bissyande T F, Thung F, Lo D, Jiang L, Reveillere L. Orion: A software project search engine with integrated diverse software artifacts. In: International Conference on Engineering of Complex Computer Systems. 2013, 242–245
31. Linstead E, Bajracharya S, Ngo T, Rigor P, Lopes C, Baldi P. P.: Sourcerer: mining and searching internetscale software repositories. Data Mining and Knowledge Discovery, 2009, 18(2): 300–336
32. Lu M, Sun X, Wang S, Lo D. Query expansion via wordnet for effective code search. In: IEEE International Conference on Software Analysis, Evolution and Reengineering. 2015, 545–549
33. Nie L, Jiang H, Ren Z, Sun Z, Li X. Query expansion based on crowd knowledge for code search. IEEE Transactions on Services Computing, 1939, 9: 1–1
34. Lv F, Zhang H, Lou J, Wang S, Zhang D, Zhao J. Codehow: Effective code search based on api understanding and extended boolean model (e). In: Ieee/acm International Conference on Automated Software Engineering. 2015, 260–270

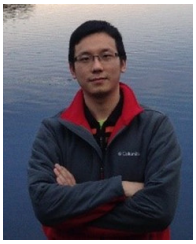
35. Mcmillan C, Grechanik M, Poshyvanyk D, Fu C, Xie Q. Exemplar: A source code search engine for finding highly relevant applications. Software Engineering IEEE Transactions on, 2012, 38(5): 1069–1087
36. Sridhara G, Hill E, Pollock L, Vijay-Shanker K. Identifying word relations in software: A comparative study of semantic similarity tools. In: The IEEE International Conference on Program Comprehension. 2008, 123–132
37. Wang S, Lo D, Jiang L. Inferring semantically related software terms and their taxonomy by leveraging collaborative tagging. In: IEEE International Conference on Software Maintenance. 2012, 604–607
38. Tian Y, Lo D, Lawall J. Sewordsim: software-specific word similarity database. 2014
39. Bhat V, Gokhale A, Jadhav R, Pudipeddi J, Akoglu L. Min(e)d your tags: Analysis of question response time in stackoverflow. In: Ieee/acm International Conference on Advances in Social Networks Analysis and Mining. 2014, 328–335
40. Wang S, Lo D, Vasilescu B, Serebrenik A. Entagrec: An enhanced tag recommendation system for software information sites. In: IEEE International Conference on Software Maintenance and Evolution. 2014, 291–300
41. Mo W, Zhu J, Qian Z, Shen B. Solinker: Constructing semantic links between tags and urls on stackoverflow. In: IEEE Computer Software and Applications Conference. 2016, 582–591
42. Chen C, Gao S, Xing Z. Mining analogical libraries in q&a discussions – incorporating relational and categorical knowledge into word embedding. In: IEEE International Conference on Software Analysis, Evolution, and Reengineering. 2016, 338–348

Zhixing Li received his BS in Computer Science from Chongqing University in 2015. He is now a MS candidate in Computer Science, National University of Defense Technology. His work interests include open source software engineering, data mining, and knowledge discovering in open source software.

Gang Yin received his Ph D degree in Computer Science from National University of Defense Technology (NUDT) in 2006. He is now an associate professor in NUDT. He has worked in several grand research projects including national 973, 863 projects and so on. He has published more than 60 research papers in international conferences and journals. His current research interests include distributed computing, information security, software engineering, and machine learning.

Tao Wang received both his BS and MS in Computer Science from National University of Defense Technology (NUDT) in 2007 and 2010. He is now a PhD candidate in Computer Science, NUDT. His work interests include open source software engineering, machine learning, data mining, and knowledge discovering in open source software.

Yang Zhang received both his BS and MS in Computer Science from National University of Defense Technology (NUDT) in 2013 and 2015. He is now a PhD candidate in Computer Science, NUDT. His work interests include open source software engineering, data mining, and social coding networks.



Yue Yu received his PhD degree in Computer Science from National University of Defense Technology (NUDT) in 2016. He is now an associate professor in NUDT. He has visited UC Davis supported by CSC scholarship. His research findings has published on MSR, FSE, IST, ICSME APSEC and SEKE. His current research interests include software engineering, spanning from mining software repositories and analyzing social coding networks.

Huaimin Wang received his PhD in Computer Science from National University of Defense Technology (NUDT) in 1992. He is now a professor and chief engineer in department of educational affairs, NUDT. He has been awarded the “Chang Jiang Scholars Program” professor and the Distinct Young Scholar, etc. He has published more than 100 research papers in peer-reviewed international conferences and journals. His current research interests include middleware, software agent, and trustworthy computing.