Linking Issue Tracker with Q&A Sites for Knowledge Sharing across Communities

Huaimin Wang, *Member, IEEE,* Tao Wang, *Student Member, IEEE,* Gang Yin, *Member, IEEE,* Cheng Yang

Abstract—Collaborative development communities and knowledge sharing communities are highly correlated and mutually complementary. The knowledge sharing between these two types of open source communities can be very beneficial to both of them. However, it is a great challenge to automate this process. Current studies mainly focus on knowledge acquisition in one type of community, and few of them have tackle this problem efficiently.
In this paper we take Android Issue Tracker and Stack Overflow as a case to study the mutual knowledge sharing between them. We propose an automatic approach by integrating semantic similarity with temporal locality between Android issues and Stack Overflow posts based on the internal citation-graph to reveal the potential associations between them. Our approach explores the internal citations in communities for closely related posts or issues clustering, exploits the rich semantics in fine-grained information of issues and posts for associations building, and leverages the temporal correlations between issues and posts in-depth for associations ranking. Extensive experiments show that the precision of our approach reaches 62.51% for top 10 recommendations when recommending Stack Overflow posts to Android issues, and 66.83% in reverse.

Index Terms—Knowledge Sharing, Citation-Graph, Semantic Similarity, Temporal Locality, Android Issue Tracker, Stack Overflow.

1 INTRODUCTION

SOFTWARE development is a knowledge-intensive activity which requires experienced developers, high-quality development knowledge and so on [1]. Traditionally, the knowledge sharing was mainly confined to closed development teams or organizations. With the globalization of software development, the whole Internet becomes a knowledge repository which consists of various communities, and the scope for knowledge sharing is largely broadened. Typically, there are two types of communities which involved in software development with unprecedented level. The first is collaborative development community such as SourceForge and Github, which provide infrastructures like version control and issue tracking services for software development [2], [3]. The second is knowledge sharing community like Stack Overflow and OsChina, which host huge amounts of comments and discussions around how software should be used, what issue users have find, which features should be provided and so on for knowledge exchanging [4], [5].

These two types of communities emphasize on different aspects of the development and application of software, and are highly correlated and mutually complementary. We take Android Issue Tracker repository and Q&A site Stack Overflow as an example. In Android issue tracker, both software developers and end-users participate in issue resolution, and form a typical development community. Stack Overflow(SO) is a prevalent Q&A site for programmers and software engineers, in which programmers post and answer questions, comment and vote on posts. Currently, SO has become one of the most predominant community for knowledge sharing around software development [6]. These two communities are interconnected. On one hand, they overlap each other as having shared participants and issues in them. On the other hand, they are mutually complementary. In Android issue tracking community, the core participants are mainly platform developers and end-users. While in android-related posts in SO, the participants are mainly Android App developers. The SO community can provide more detailed information for issues from different perspectives by these huge amounts of App developers. Meanwhile, the dynamics and the platform developers' discussions of the issues in issue tracking community will help SO users to identify root cause for their questions, keep track on the progress and solve their questions.

To mostly leverage the potential of the scattered knowledge in the two types of communities, it is crucial to reveal the associations between them and link them to bridge the communities for knowledge sharing. However, as different communities have different mechanisms and emphasises, and the knowledge is expressed in informal free texts, automating this process is a non-trivial problem.

Recently, some preliminary researches have been conducted on retrieving useful knowledge from various communities. Bacchelli et al. [7] devised an approach to link high-level design decisions in emails with low-level implementations in source code for software comprehension. Subramanian and other researchers [8], [9] proposed different methods to retrieve relevant learning resources or usage examples for APIs to help developers when programming. Most of these works are code-centric which focus on linking external resources with source code artifacts. In addition to these works, many studies have been done on utilizing the crowdsourced knowledge in SO. Bacchelli et al. [10] developed a plugin to combine SO with Eclipse to facilitate the utilization of SO knowledge when programming. Correa

• *Huaimin Wang, Tao Wang, Gang Yin and Cheng Yang are with the Key Laboratory of Parallel and Distributed Computing, College of Computer, National University of Defense Technology, Changsha, China, 410073. E-mail: {hmwang, taowang2005, jack.nudt, delpiero710}@nudt.edu.cn*

Manuscript received xxx xxx, 2015; revised xxx xxx, 2015.

et al. [11] explored the influence of introducing SO posts to Android and Chromium issues and find it valuable for issue fixing. These works aim at unidirectional knowledge acquisition from Stack Overflow, and little attention has been paid to mutual knowledge sharing between issue tracking communities and Q&A sites.

In this paper we take Android Issue Tracker and Stack Overflow as a case to study the mutual knowledge sharing between them. We propose a novel automatic approach which integrates internal citations, semantic similarity with temporal factors to mine potential associations between them for automatic knowledge sharing. Different from the previous work, we study the problem from both directions: recommending relevant issues to SO posts and the reverse, and build a uniform similarity model to accomplish this process. Specifically, we cluster the posts or issues based on the internal links, and then analyze the capability of different IR approaches at retrieving semantically relevant issues or posts clusters with multiple granularities of information. As correlated issues and SO posts are rooted in the same issue, the time they arise in Issue Tracker and SO should be temporally related. We explore the inherent temporal correlations between correlated issues and posts and combine them with semantic similarity to optimize the recommendation performance.

In addition, based on the basic idea we build *OSSEAN*, an *O*pen *S*ource *S*oftware *E*valuating, *A*nalyzing and *N*etworking Platform. *OSSEAN* links application data like user feedbacks in knowledge sharing communities to corresponding projects in collaborative development communities, and provides an integrated platform for exploring these projects comprehensively from the perspectives of both developers and users.

The main contributions of this paper are as follows.

- The mutual benefits of knowledge sharing between Android Issue Tracker and Stack Overflow are explored from different perspectives, to which attention has been rarely paid before.
- A novel approach combining internal citations, semantic similarity with temporal-locality is proposed, and a uniform similarity model is designed for recommending issues to posts and the reverse. The internal citations and the temporal locality are never considered before in the previous works.
- Three typical IR approaches are investigated and different granularities of text information for issues and posts are explored for recommendation.
- Comprehensive experiments are conducted and different approaches are compared which demonstrate the validation of our method.

The remainder of this paper is organized as follows. Section 2 analyses the benefits of knowledge sharing between Android Issue Tracker and SO. Section 3 describes our approach in detail. Section 4 presents the research questions and experiment settings and Section 5 evaluates our approach with extensively experiments. Section 6 discusses the validity of our work. We review the related work in 7 and summarize this paper in Section 8.

2 Benefits of Knowledge Sharing between Android Issue Tracker and Stack Overflow

With the globalization of software development, knowledge sharing is no longer confined within an organization or a team, but broadened to cover various types of software communities [12]. In this section we take Android Issue Tracker and SO as a case to explore the benefits of knowledge sharing between them.

2.1 Data from Android and Stack Overflow

In Issue Tracker users can submit issue reports or participate in corresponding discussions. An issue report and its responses are organized into a thread which we call an *issue thread*. So is that in SO where users post questions or answer and comment them. A question and its answers and comments are grouped into a thread which we call a *post thread*. To analyze the potential benefits of knowledge sharing between these communities, we focus on the issue threads from Android Issue Tracker and the post threads in SO. The details are presented in Table 1.

TABLE 1
Statistics of Experiment Datasets

Community	#Total posts and answers	#Total threads	#Threads with links	Time period
Android Issue Tracker	151,815	30,572	653	2007-11-12 ∼ 2013-09-02
StackOverflow (Android)	1,169,415	387,422	2,753	2008-07-31 ∼ 2013-09-06

For Android issues, we crawled all its issues from the first reported issue in Nov. 2007 to the ones until Sept. 2013[1]. While for Stack Overflow, it has attracted huge amounts of software developers asking and answering programming-specific questions since its launch. In this paper, we use the publicly released dataset for SO[2], which contains all the questions, answers, comments and other information from Jul. 2008 to Sept. 2013. And we only focus on the post threads which are tagged with *Android*, which discuss questions relevant to Android platform.

In the raw dataset, there are more than 50,000 issue threads, we exclude those threads whose status are “Decline","Spam", “WorkingAsIntended", “Unreproducible", “WrongForum", “NotEnoughInformation" or “UserError". We filter such issues which are viewed as invalidate and get 30,573 issue threads including 151,815 issues and answers. Among these threads 653 ones contain links to SO question threads, either in the issue or in the answers. In SO, the total number of the questions and answers tagged with “android" reaches 1,169,415, and they are organized into 387,422 threads. Among these threads 2,753 ones contain links to android issues. Based on the selected data we analyze the characteristics of links between these two typical communities.

2.2 Benefits of Knowledge Sharing for Android Issue Tracker

The traditional Issue Tracker mainly focus on collecting user feedbacks and tracking issues. Its participants include

1. https://code.google.com/p/android/issues/list
2. https://archive.org/details/stackexchange

platform developers and users, and the active participants are mainly a small group of core developers. While for SO, the majority of its participants in Android-related posts are the platform users who are developing applications for the Android platform. When confronted with programming-specific problems they can discuss and share their experiences in SO. Table 2 compares the statistics of participants and responses in Android Issue Tracker and SO.

TABLE 2
Users and responses in Android Issue Tracker and SO

Community	#Total users	#Active users	#Average responses	Median time for first responses
Android Issue Tracker	58,112	525	4	31(*days*)
Stack Overflow	171,152	18,205	5	13(*minutes*)

From Table 2 we can see that the total number of distinctive participants in Android-related posts in SO is almost three times of that in Android Issue Tracker. Among them, the number of active participants (who participates in discussions in more than ten issue or post threads) in Android Issue Tracker is only 525; While in SO there are more than 18,000 ones, which is much more than that in Issue Tracker. These active participants, as experienced developers, are of great value for bug fixing as they can provide more detailed information or possible solutions from different perspectives.

Besides, we compare the responding time in these two communities. We count the time interval between the issues (or SO questions) and their first responses. For those issues and posts which have no response yet, we assume the time of their responses as *2013-09-06* which is the last day of the data record in our dataset. As shown in the last column of the table, the median time of first responses for Android issues and SO posts differs greatly. The median time in SO is only 13 *minutes*, while in Android Issue Tracker it is 31 *days* which is much longer. Issue fixing can benefit much from the quick response in SO. Furthermore, the quick response will motivate corresponding developers/users to become long term contributors, which is crucial for OSS success [13].

In short, by linking related SO posts to issues, Android Issue Tracker can leverage the huge amounts of external developers and quick responses for issue fixing. In recent years, more and more developers realize the benefits of SO discussion posts for issue resolution, and they introduce related posts to issues manually. Figure 1 presents such growing trends in recent years in Android issue repository.

In the beginning, only few issues contain links to SO posts. This is mainly due to that the number of issues submitted are not so large and SO was only launched for a few months. As time goes on, more and more Android developers begin to include links to corresponding SO posts in their discussions to pull external knowledge to Issue Tracker. Although there are ups and downs, the overall curve shows an upward tendency.

Among these links, some are included by developers who happen to know that there are related posts in SO; Some others are pull-in by developers who post the corresponding questions in SO consciously to inspire discussions over the issues in SO. We further compare the number of comments for issues with and without SO post links. We

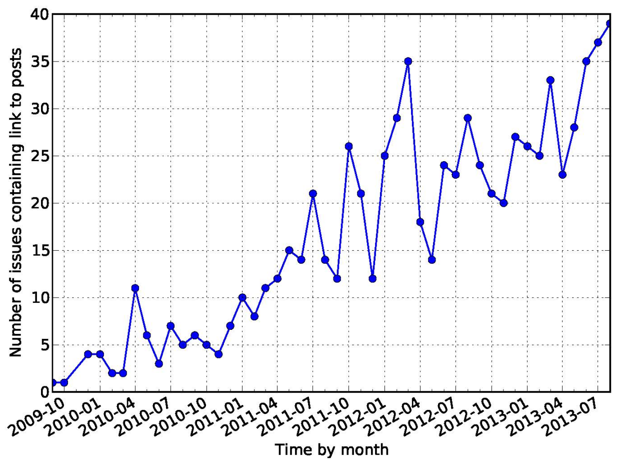


Fig. 1. The growing trend of new arisen Android issues containing links to SO posts from Oct. 2009 to Sept. 2013

find that the average number of comments for issues having post links is about 6.87. While that for issues without post links are only 3.85, which is much less than that having SO links. This suggests that these links introduce the external knowledge in other communities and inspire more extensive and intensive discussions in Android Issue Tracker. This will obviously be helpful for issue resolution.

2.3 Benefits of Knowledge Sharing for Stack Overflow

For SO, the related issues in Issue Tracker can be helpful for answering programmers' questions. Most active participants in Issue Tracker are mainly professional system developers who are quite familiar with the corresponding modules. Thus, if a given question in SO is associated with an existing issue, the discussion of the issue will make questioner and viewers clear about the cause and solutions to the question. This helps to solve their problems. Currently, in SO there have been a lot of android-related posts which contain links to issues.

In SO a question can receive several answers which are organized into a thread, and the questioner can choose one answer which he thinks the best to his question as *accepted*. Besides, the registered viewers can vote up or vote down the answers based on their judgements. Table 3 presents such information of these post threads which contain issue links.

TABLE 3
Statistics of SO posts with links to issues

	Number	Ratio
Post threads that contain issue links in answers	2,378	—
Post threads that have accepted answers	1,512	—
Accepted answers that contain issue links	1,021	**67.53%**(1021/1512)
Mostly accepted answers that contain issue links	1,601	**64.74%**(1601/2378)
Voted as top-2 answers that contains issue links	2,121	**85.77%**(2121/2378)

In SO dataset, there are 2,378 android-related questions which contain links to issues in answers. There are 1,512 questions having *accepted* answers, among which 67.53% of the questioners choose those answers containing issue links as the *accepted* ones. This suggests that the majority of the questioners view these answers as best. In addition, from the viewers' perspective, 64.74% of the mostly voted-up answers are those containing issue links, and that for answers voted up as top-2 reaches 85.77%. From the acceptance and vote-up information, the majority of the participants view those answers containing links as the most satisfying ones. This indicates the value of including links to issues in post discussions for solving programmers' questions.

Besides, we analyze the number of posts containing issue links monthly from Jan. 2009 to Sept. 2013. Figure 2 presents such growing trends in recent years in SO community.

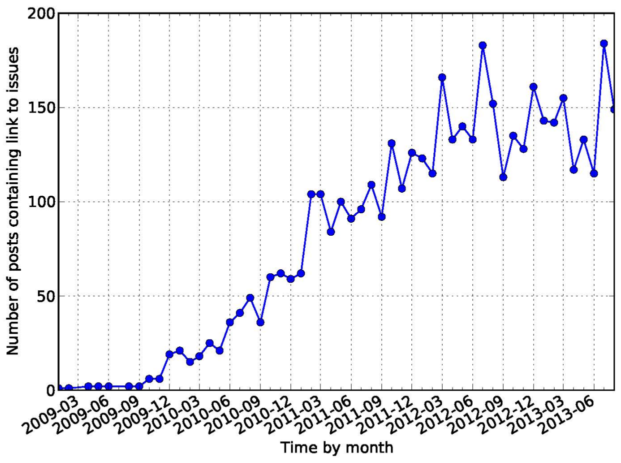


Fig. 2. The growing trend of new arisen SO posts containing links to Android issues from Jan. 2009 to Sept. 2013

Just as what are happening in Android Issue Tracking community, more and more programmers introduce related issues to posts manually at answering questions. From this figure we can see similar growth as that in Android Issue community. From Mar. 2012, the number of posts containing issue links increases to about 150.

In summary, the statistics of these two communities suggest that both communities can benefit much from each other. However, currently links are added manually by developers, and they only take a small proportion of the whole issues and posts. These motivate us to mine potential associations between issues and posts automatically for bidirectional knowledge sharing.

3 Our Approach

In this section, we firstly present the overview of our approach. We explore the internal links in open source communities and propose a citation-graph based clustering approach. Then we discuss the granularity of information used for retrieving, and analyze the semantic similarity based approaches. In the end we explore the temporal correlations between related issues and posts and propose a novel approach combining semantic similarity with temporal factors to retrieve correlated issues and posts.

3.1 Overview of Our Approach

To reveal potential associations between Android Issue Tracker and SO, we focus on two factors: text similarity and temporal correlation. The more similar two threads, the higher the probability they are correlated. Meanwhile, for two threads in different communities which are rooted in the same issue, the time they arise should be in a same short period of time. This can be viewed as a kind of *temporal locality*. Based on these intuitions, we propose an approach which synthesizes the semantic similarity with temporal locality to find correlated threads across communities. The overview of our approach is illustrated as Figure 3.

Our approach consists of four stages, including *Data Extraction*, *Citation-graph based Clustering*, *Model Training* and *Links Recommendation*.

In the *Data Extraction* stage, three types of information are extracted from the Android issues and SO posts. The first is the text contents including the title and tags of issues/posts, issue/post contents and corresponding answers. For a given issue in Android Issue Tracker or a question post in SO, the issue/post and the corresponding answers are organized into issue thread or post thread. We treat the text contents of a issue/post thread as a whole to represent the issue/post. The second is the internal links in SO posts which are added by questioners or answerers to refer to other related posts. The third is the temporal attributes, including the reporting and responding time of issue and post threads. These types of information will be used for clustering and model training.

In the *Citation-graph based Clustering* stage, posts in SO are clustered. In SO, there are many links in posts which refer to other related posts. By employing these internal links, we cluster closely related posts.

In the *Similarity Model Training* stage, post clusters, issues and their temporal attributions are leveraged to build similarity models. The extracted thread texts are used to build the Semantic Similarity Model, and the temporal attributes are employed to build the Temporal-Locality Model. When building the semantic and temporal models, the post clusters and issues instead of each single post thread are used. Then these models are integrated as the Synthesized Similarity Model.

In the *Links Recommendation* stage, the thread texts and temporal attributes of a query issue or post are used as the input of the Synthesized Similarity model. Based on the synthesized similarity between the issues and the post clusters, a list of post clusters or issues will be returned as potential associations. The details are described in the following sections.

In data extraction stage, the texts of issues and posts can be divided into different granularities including titles, descriptions and answers. Different granularities of texts may affect the performance for similarity models training in the next stage. The temporal attributes include the time the issues reported and the SO questions raised which we call *Reporting-Time* and the time the corresponding responses appear which we call it *Responding-Time*.

In the third stage, we train semantic similarity model based on thread texts and build temporal locality model based on temporal attributes, and then synthesize them together as a integrated model. Many different IR approaches can be employed here to build the semantic similarity model. The details of the semantic similarity model and temporal locality model are presented in the following two subsections. At the fourth stage, for a given query of issue or post, we recommend candidates to users according to the synthesized similarities.

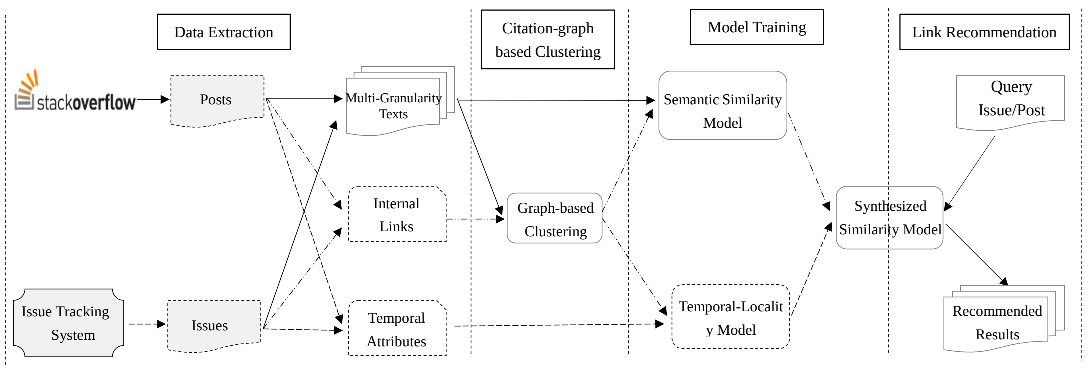


Fig. 3. The overview of our approach

3.2 Citation-Graph based Clustering

In Stack Overflow, different users may ask similar or related questions, and participants may introduce links in answers to refer to other related posts in the community manually. These internal links, which connect related posts that focus on similar or related questions, form inter-connected knowledge graph [14]. In Android Issue Tracker we can see that there are also links refer to related issues. These connected posts or issues cover different aspects of the core question and give more comprehensive information about it. So we firstly cluster these closely related issues and posts separately based on these internal links. Then we can calculate the similarities between these clusters instead of single issues or posts.

These internal links in Stack Overflow and Android Issue Tracker form a kind of weighted undirected graph which models the citation and semantic correlations between post threads or issue threads. Take that in Stack Overflow as an example. We define the weighted citation graph in Stack Overflow as $G =< V, E, W >$. In this graph,

$$G(V) = \{v_i | \forall v_i \in threads\} \quad (1)$$

where $threads$ is the set of post threads which are tagged with “Android" in Stack Overflow.

$$\begin{aligned} G(E) =& \{(v_i, v_j) | \forall v_i, v_j \in G(V) \\ & \& \ (v_i \rightarrow v_j \ or \ v_j \rightarrow v_i)\} \end{aligned} \quad (2)$$

where $\rightarrow$ represents the “cite" relation between two post threads. The weight of a given edge $w(v_i, v_j)$ is defined as the semantic similarity between two connected vertices v_i and v_j which can be acquired by using Cosine Similarity.

In this graph, there can be isolated nodes which stand for post threads that do not cite or cited by any other posts. For Android Issue Tracking community, we can build similar citation graph. The only difference is that the vertices are the issue threads. Figure 4 presents a connected component example in SO citation-graph.

Among the edges(the internal links) in the constructed graph, a proportion of them may refer to other topics which are not so coherent to the core question. To find the closely related nodes, we focus on two metrics to divide large connected components into small isolated but coherent clusters. The first is semantic similarity and the second is the diameter of the connected components.

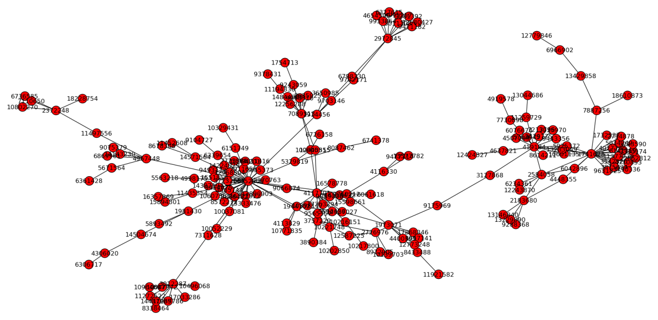


Fig. 4. An example of connected component in SO citation-graph

Firstly, we focus on the weights of edges in the graph which represent the degree two nodes are semantically related. For two connected nodes, we can determine if the topics two nodes discuss are coherent or not by analyzing the semantic similarity between them. If the semantic similarity is under a certain threshold, we can view them as not closely related and the corresponding edge in the graph will be deleted. The semantic similarity only measures the coherency between two directly connected nodes.

Secondly, we analyze the diameters of the connected components in the graph. For a given path in the graph, even the semantic similarities between the pairs of directly connected nodes are all above the similarity threshold, the focus of two nodes which are connected through many intermediate nodes can differ a lot. This is similar to the spread of influence in social network. We use the diameter as a metric to depart the large connected graph into ones with diameter smaller than a given threshold.

3.3 Semantic-Similarity Based Recommendation

Semantic similarity is the basic measure to identify if two threads are correlated. In this subsection we discuss the granularity of text information available and the Information Retrieval approaches we use for semantic similarity analysis.

3.3.1 The granularity of information used for measuring semantic similarity

In Issue Tracker and SO, the issues, posts and the corresponding answers and comments are organized into threads. A thread often contains title, tags, issue/question description, answers and comments, which provide different granularities of information and thus affect the linking accuracy. The titles and tags give high-level summarization and core concepts. Such texts are often written carefully and of high quality. However, due to the length constrain they cannot cover all aspects of the issue or question. Descriptions of issues or questions provide more details and are important for identifying concrete problems. For answers given by different users, they present additional useful information, while at the same time introduce more noises and transfer to other topics [15]. In addition, in SO users can comment the questions and answers which discuss some tips.

In this paper we emphasize on building links between the Android Issue Tracker and SO. Due to the prevalence of the Android platform, both of these two communities have attracted a huge number of users as shown in Table 2. The vocabularies used by these different users and the writing styles differ much, which increases the difficulty when calculating the semantic similarities. Due to this, it is important to include the answering information for semantic similarity analysis, even these answers may introduce more noises. Because the answers are from different users, the inclusive of them can broaden the vocabulary for the threads and enlarge the intersections between the issue and post threads. While for the comments in SO, we neglect them because most of them contain many specific details and noises.

To take full advantage of the fine-grained thread information and meanwhile reduce the influence of noises, we adopt several strategies to select highly topic-sensitive terms for similarity calculation. Firstly, we eliminate those terms that appear seldom and too frequent in the whole dataset like “android" which are of little value at distinguish two texts in the corpus. Secondly, we eliminate those texts with only a few words because most of them are simple comments and contain little information about the issues or SO questions. Thirdly, in the descriptions of issues or questions there are often long error trace information which contain a lot of noises. For such snippets we only retain the code-like terms which reflect the code-related information of the issues or questions. After these steps, we integrate the multiple granularities of information including title, tags, descriptions, answers to represent the corresponding issue and post threads.

3.3.2 *Semantic-Similarity Based Approaches*

To compute the semantic similarity between two texts, Cosine Similarity is a widely used approach [16], [17]. It is a similarity measure between two vectors by calculating the cosine of the angle between them. To calculate the Cosine similarity between two texts, the first step is to represent the texts with weighted vectors. Then it can be calculated with Equation 3.

$$CosineSim(V_i, V_j) = \frac{\sum_{k=1}^{|V|}(V_{i_k} \times V_{j_k})}{\sqrt{\sum_{k=1}^{|V|} V_{i_k}^2} \times \sqrt{\sum_{k=1}^{|V|} V_{j_k}^2}} \tag{3}$$

where V_i, V_j are the document vectors and V_{i_k} is the kth element in the vector. The value of V_{i_k} represents the importance of the kth term for the the ith document.

There are many techniques to represent documents with weighted vectors. Three typical models Vector Space Model(VSM), Latent Semantic Indexing(LSI) and Latent Dirichlet Allocation(LDA) have been used in different software engineering tasks including API methods retrieving [18], [19], similar applications finding [17], [20] and code-related documents analysis [5], [15]. All these models view documents as bag of words without considering the sequences they appear in the texts. The main difference between them is how they map the documents into a new space.

Vector Space Model. The Vector Space Model represents each document as a weighted vector, in which each element is a term and the value stands for its importance to the document. The weights can be measured by the times it appears in the document and the number of documents it appears. TFIDF synthesizes these two together and is commonly used to calculate the term weights. We adopt it in this paper as well which can be calculated as formula 4.

$$TFIDF(t, d, D) = \frac{t_d}{|d|} \times \log \frac{|D|}{N_{tD}} \tag{4}$$

where t is a term, d is a document and t_d is the times t appears in d. D is the corpus consists of multiple documents, and N_{tD} represents the number of documents in which the word t appear.

Latent Semantic Indexing Model. Because of the synonym and polysemy in natural language, different people may express the same meaning with different words or mean different using the same word. Latent Semantic Indexing(LSI) [21] elevates terms to an abstract space, and synonym terms used in similar contexts are considered similar even they are spelled differently. Instead of focusing on terms, LSI reveals latent concepts expressed in documents and captures most essential semantic information. LSI makes use of singular value decomposition to represent each document with a weighted vector. In the vector each element is a concept embedded in the document, and the weight represents the importance of the concept to corresponding document.

Latent Dirichlet Allocation Model. LDA is an unsupervised topic model. The basic idea of LDA is that documents are random mixtures over latent topics, and each topic is a distribution over words [22]. Given a corpus of documents, the topic number K, LDA can infers a set of K topics over the whole set of documents. LDA represents each document as probability topic vectors, in which each element is a topic and the value stands for the probability of that topic for expressing the document. Compared with LSI, it is also a topic model, but it can alleviate model overfitting problem.

3.4 Temporal-Locality Based Recommendation

3.4.1 *Reporting-Time Locality*

As one of the most popular mobile OS, Android has attracted hundreds of thousands of application developers to develop Apps for it(As presented in Table 2, there are 58,112 participants in Android Issue Tracker and 171,152 ones in SO). Thus, if defects are exist in the platform which result in problems, they will be perceived quickly by the large number of users. Some users report it in Android Issue Tracker, and some may post a question in SO. As the correlated issues and posts are caused by the same defect in Android system, the time the bug report arise in

Issue Tracker and that the question posted in SO should be temporally related, which we call *temporal-locality*.

The Reporting-Time Locality is based on the intuition that the discussions in different communities inspired by the same root cause should arise in a short period of time. Such temporal-locality captures the inherent correlations between the related posts and issues.

To gain insight into the temporal correlations between them, we analyze the distribution of the reporting time interval over the correlated issues and questions based on the data presented in Table 1 in Section 4.

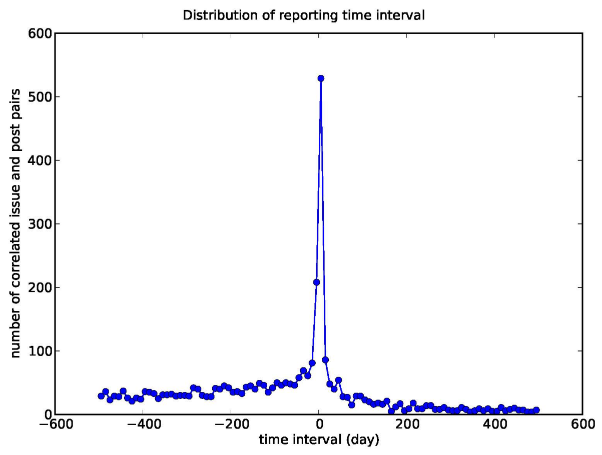


Fig. 5. The distribution of reporting time intervals between explicitly linked issues and posts. Negative time intervals suggest that the issue arises earlier than the corresponding post.

Figure 5 presents the details of the distribution. The x axis is the time interval in days, and the y axis is the number of correlated pairs which are of that time interval. To measure the time interval, we use reporting time of the bugs in Issue Tracker minus that of the correlated questions in SO. If a post arises later than the correlated bug report, then the time interval between them will be negative. In addition, we divide the time intervals into ranges of [-10, 0), [0, 10), [10, 20) and so on with unit of days, and represent them with the medians at the x axis, like 5 for [0,10). Then we count the number of pairs in these ranges.

As shown in this figure, the number of pairs within [0, 10) is of the most which is more than 500, and that within [-10, 0) is more than 200. Specifically, among all the issues that having link to SO posts, about 75.5% of them are firstly raised in SO and then been discussed in Android Issue Tracker, in which about 15.6% are discussed in the same day. The remaining 24.5% are firstly raised in Android and then discussed in SO. For this proportion, they are firstly submitted to Issue Tracker. After several days of discussion, the programmer in Android may find related discussions about the defect in SO, then he add the corresponding SO posts in the discussion of that issue in Android.

Overall, the majority of the correlated pairs arise in a period of 60 days, which suggests the temporal correlations between correlated issues and questions. Overall, the total number with negative values are larger than that with positive values. This is because that the number of correlated pairs in SO dataset (containing issue links in the post thread) is larger than that in Issue dataset (containing post links in the issue thread). If taking this into consideration, the positive and negative ones should be close to each other.

3.4.2 Responding-Time Locality

In addition to the reporting-time locality, there is another kind of temporal locality over answering/commenting time. Such temporal locality is attributed to the shared developers who participate in related activities in the two communities.

In open source communities, quite a large number of developers are very active and participate in different communities, and their activities in these communities are coherent [23]. For a given developer, the issue/question he pays attention to at a given time is heavily related to what he is doing in hand right then, and what he pays attention to in two communities should be temporally close.

Issue 57820[3] in Android Issue Tracker is an example. *Michael* faced a problem when he updated his Android Studio and he posted a question in SO to seek help. One day later he discovered an open issue in Android Issue Tracker. Then he commented the issue with a link to his post. Inspired by this example, if we can identify such developers who appear in two communities, we can take advantage of it to find highly related issues and posts. This is based on the hypothesis that a pair of similar issue and post which have shared participants should have higher possibility of being correlated than others that have no shared participant.

However, Android Issue Tracker adopts a strict privacy mechanism, which makes it unable to identify such developers automatically in the communities. Nevertheless, a developer's attention on a specific topic would have temporal locality as shown in the aforementioned example. Based on this intuition, if a post and an issue are similar in semantic and at the same time the responding time are close, we can assume that the responses may come from the same developer, and thus raise the probability that the post and issue are correlated. This is another type of temporal locality we can utilize.

We analyze the distribution of the reporting and responding time interval over the correlated issues and questions based on the data presented in Table 1 in Section 4. We find that the majority of the correlated pairs arise in a period of less than 30 days, which suggests the temporal correlations between correlated issues and questions.

3.4.3 Temporal Locality based Approach

To take advantage of temporal locality, we propose a novel approach which synthesizes the semantic and temporal factors at identifying related issue and post threads. Semantic similarity is the basis of the approach. For those issues and posts which have similar level of semantic similarity, if the reporting time or responding time is close, the probability that they are correlated will be higher than that appear with large time interval. Based on this, we design the following Equation to integrate semantic similarity with temporal locality.

$$Sim(ij) = \alpha \times Sim_{content}(ij) + \beta \times Sim_{temporal}(ij) \quad (5)$$

where

$$Sim_{temporal}(ij) = \frac{1}{\min(time_{intervals}(ij)) + 1} \quad (6)$$

3. https://code.google.com/p/android/issues/detail?id=57820

In Equation 5, $Sim_{content}(ij)$ and $Sim_{temporal}(ij)$ stand for the similarity based on $issue_i$ and $post_j$(or $post_i$ and $issue_j$) contents and reporting/responding time respectively with unit of "day". α and β are the weight factors. The sum of α and β should be 1 and their values can vary between 0 and 1. If $\alpha = 1$ and $\beta = 0$, then it means only considering the semantic similarity and without taking temporal locality into account, and vice versa.

The $Sim_{content}(ij)$ can be calculated with IR techniques discussed in Section 3.3.2, while $Sim_{temporal}(ij)$ can be calculated with Equation 6. $time_{intervals}(ij)$ is a set of time intervals between the issue and post thread i and j, including the reporting and responding time intervals. We do not differentiate the reporting time and responding time. In the resulted time interval set, we only consider the min interval.

4 Experiment Settings

In this section, we introduce the research questions, the dataset and the evaluation metrics.

4.1 Research Questions

The experiments aim to address the following research questions:

- **RQ1:** Which Information Retrieval approach is most effective for linking issues and posts?
- **RQ2:** What granularity of thread information is most effective for linking issues and posts?
- **RQ3:** Whether or not temporal locality could improve the linking accuracy?
- **RQ4:** Can citation-graph based clustering improve the linking accuracy?

Many IR techniques have been proposed for retrieving. RQ1 aims to compare the performance of three widely used techniques at finding related issues and posts. In Android Issue Tracker and SO, these are different granularities of information be available for retrieving. RQ2 analyzes what granularity is most effective. For correlated issues and posts, as they are rooted in the same defect, the reporting or responding time in both communities should be correlated. RQ3 aims to see if the proposed temporal-locality based approach is effective at improving the linking accuracy. There are many internal links in Stack Overflow, based on which we propose to use these links to clustering and linking, and RQ4 means to reveal the validation of this approach.

4.2 Experiment Dataset

Android Issue Tracker and Stack Overflow are typical development community and knowledge sharing community. To validate our approach we focus on the dataset illustrated in Section 2.1. As there are links which are added by developers or users, in the experiments we train a uniform similarity model using all the issue and post threads, and use these already existing links as the ground-truth to test.

For text mining, the text preprocessing is very important. In issues and posts there are quite a lot of noises which affect the final performance. In this paper we mainly take the following steps to preprocess these texts, including stop word removing and word stemming. At stop word removing, words like "a","the" are filtered out. For word stemming, we use Porter Stemmer to transform words to their stems to eliminate the differences like singular or plural form. Besides, we set upper limit as one third percent of the total texts and the lower limit as 5 times to remove those words that appear too frequent or seldom in the corpus.

4.3 Evaluation Metrics

Precision and Recall are widely used metrics for evaluating recommendation performance. In this paper, we chose those issue threads (or post threads) which contain links to posts (or issues) as the testing dataset, and view the existing links as the standard answers. For most testing data only have one link in each of them. So in this paper, we do not measure the recall but only precision.

For recommending results, the users may only concern the top ranked results and neglect the results ranked far behind. So in this paper we only consider the top-k ones. For a given query, if the correct answer is contained in the top-k recommendations then we will view this recommendation as correct. Otherwise, it will be viewed incorrect. We calculate the performance over all the queries and get the Mean Precision for recommending K(**MP@K**). In addition, the rank of the correct answer in the top-k results is another important metric. The higher the rank, the better the result. Here we use Mean Reciprocal Rank over K(**MRR@K**) to evaluate the average rank of the correct answers for all queries. The detailed definition of these two metrics are as follows.

$$MP@K = \frac{1}{|Q|}\sum_{i=1}^{|Q|} result_i \qquad (7)$$

$$MRR@K = \frac{1}{|Q|}\sum_{i=1}^{|Q|} \frac{1}{Rank_{K_i}} \qquad (8)$$

In Equation 7 and 8, Q means the set of query samples, and $result_i$ stands for the value of recommendation for query i. If the correct answer appear in the top-k results, then the value is 1, elsewhere 0. $Rank_{K_i}$ in Equation 8 represents the rank of the correct answer in the top-k recommendations. For query i, if the correct answer appear in top-k results, then $Rank_{K_i}$ is its rank. Otherwise, its rank will be viewed as infinite, which make the reciprocal as zero.

5 Results and Discussion

In this section we mine the associations between Android Issue Tracker and Stack Overflow for knowledge sharing from both directions. We train two uniform similarity models: the first is built based on all Android issues except those including SO post links and we use this model to recommend posts to issues. And the second is built based on all Stack Overflow posts tagged with "android" except those including issue links and we use this model to recommend issues to posts. We measure the average precision and MRR over top 5, 10, 20 and 50 results(10 is the number of results presented in the first page in popular search engines like Google and Bing). In addition, to see the possible best results we may achieve, we also show the performance over top 100, which provides us a target to optimize our approach.

To address the research questions in Section 4.1, we conducted extensive experiments. Firstly, we compare the effectiveness of VSM, LSI and LDA at finding related post or issue threads based on text contents. Then, we analyse the influence of thread granularity for recommendation. Finally, we evaluate the synthesized similarity model.

5.1 RQ1: Different Information Retrival Approaches

Semantic similarity is the basis for identifying possible associations between issues and posts. In this subsection, we compare the VSM, LSI and LDA models at finding correlated issues and posts based on the texts of titles, tags and issue/question descriptions.

For LSI and LDA, we need to predefine the parameters of topic number and iteration times. Considering the huge amounts of issues and posts and the large scale of the Android system, we set the topic numbers as 300 and the iteration times as 1000. The other parameters are set to their default values. Table 4 and 5 present the detailed results.

TABLE 4
Using VSM, LSI and LDA to recommend posts to issues

Approach	Top 5		Top 10		Top 100	
	MP	MRR	MP	MRR	MP	MRR
VSM	**0.3231**	**0.2509**	**0.3675**	**0.2571**	**0.5636**	**0.2644**
LSI	0.2695	0.2101	0.3017	0.2142	0.4579	0.2198
LDA	0.0475	0.0344	0.0551	0.0351	0.1164	0.0372

TABLE 5
Using VSM, LSI and LDA to recommend issues to posts

Approach	Top 5		Top 10		Top 100	
	MP	MRR	MP	MRR	MP	MRR
VSM	**0.3713**	**0.2761**	**0.4444**	**0.2859**	**0.6963**	**0.2950**
LSI	0.2235	0.1614	0.2739	0.1682	0.5319	0.1774
LDA	0.0968	0.0634	0.1264	0.0673	0.3282	0.0741

Comparing the three techniques based on precision we can see that VSM performs best. Taking retrieving related issues for SO posts as an example. Over all testing set VSM can get the correct result in the top-5 recommendations for 37.13% of the queries. While for LSI and LDA, they can only achieve 22.35% and 9.68% respectively. Comparing the average rank of the correct links, the MRR for VSM is about 0.25 for recommending 5 candidates, which means that the average rank of correct results over all testing data is about 4. While for LSI and LDA the MRR are only 0.2101 and 0.0634. The reason why VSM performs better than LSI and LDA mainly due to the characteristics of the issue-related contents.

Although the topics of an issue or a post mean to be concentrate on a specific function or module, the description of issues or questions often introduce other topics and have low coherence [15], [24]. Such dispersity affects the performance of LSI and LDA when calculating the similarity. Besides, as these issues and posts focus on the specific platform of Android system, quite a lot of words are commonly used in even irrelevant issues and posts. The key concepts for a given issue or post are often subtle and reflected by a few keywords. When mapping to greatly reduced dimension of topic space, LSI and LDA fails to perceive such subtle dissimilarity, especially for the short texts of issue/post descriptions. While VSM keep all the original terms without lossing any information but distinguish those keywords with TF-IDF.

In addition, from Tables 4 and 5 we can see that, for all three IR approaches the results for recommending issues to posts are better than that for recommending posts to issues. This is due to the quality difference between issue reports and SO posts. In SO, the gamification mechanisms motivate participants to provide high-quality contents, and questioners who want to get quick response would pay much attention to describe the question precisely. While for Android Issue Tracker, although there are guides for bug reporters, they are not motivated enough to pay much attention to the issue report quality [25], which results in low-quality reports. When using a single issue as a query, its low quality will affect the performance. The searching scope is another influential factor. When retrieving related posts, the searching scope is more 380,000 post threads. While retrieving related issues, the scope is only about 31,000 issue threads.

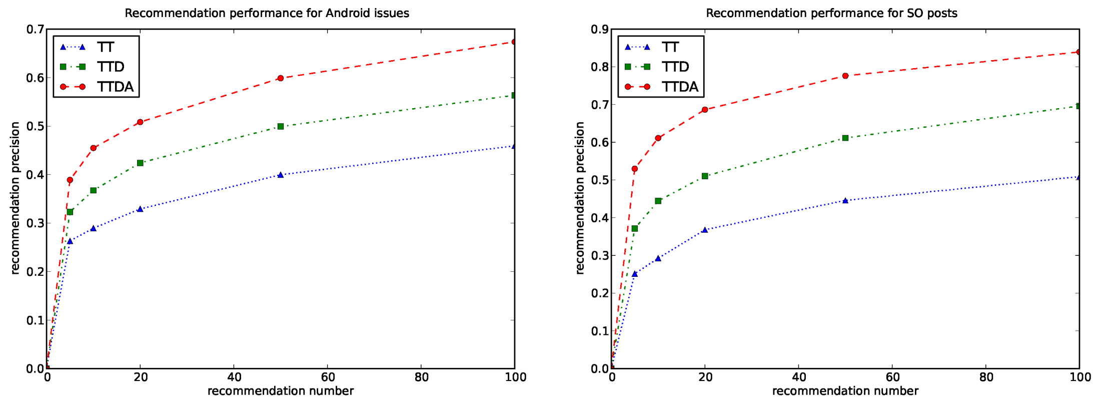
5.2 RQ2: Different Granularities of Thread Information

For a given issue or post, the title and tags cover the core concepts and give a summary of the full thread. The descriptions and discussions reveal more details. However, the fine-grained information may also introduce more noise or even transfer to other topics [15]. In this subsection, we employ different granularities of information to build semantic similarity model and do recommendation to see how the granularity affect the performance.

The experiment in Subsection 5.1 suggests that the VSM approach performs best. Due to the page limitation, in this subsection we only present the results using VSM over different granularities of thread information. We use *TT* to stand for the texts of **T**itle plus **T**ags, *TTD* for *TT* plus **D**escription of bug report or SO question, and *TTDA* for the most detailed information including *TTD* and the corresponding **A**nswers. We test and evaluate the performance for recommending 5, 10, 20, 50 and 100 candidates in this subsection. The detailed results are shown in Figure 6.

For both experiments, the more fine-grained information used, the higher the recommendation precision. Compared with the previous work [11] which only use title and tags to calculate semantic similarities, our approach takes issue/question descriptions and corresponding discussions into consideration and can significantly improve the precision. Taking Figure 6(b) as an example, the precision increases from 25.16% to 53.00% when recommending top 5. Similar improvement can be achieved for recommending posts to issues. This experiment proves that the proper use of fine-grained information can significantly improve the recommendation performance.

One question authors may ask is that, for new submitted issues or posts, there maybe not too much discussion information. For these issues/posts the performance will decrease. For newly submitted issues/posts, we can still use the issue/question descriptions in addition to the titles and tags. As shown in Figure 6, the precision by using *TTD* can also achieve more than 40.00% for both experiments.



(a) Recommending SO posts to Android issues (b) Recommending Android issues to SO posts

Fig. 6. The performance of recommending different number candidates based on *TT, TTD and TTDA*

Another thing needs to note is that, in Issue Tracker and SO lots of issues and posts are duplicate or closely related. However, to automate our evaluation, in this paper we only view these manually added links in issues or posts as the correct answers, and the others even though they are the duplicate of the standard answers will be viewed as incorrect. For some queries, although the standard answers are not contained in the top-k recommendations, the duplicate or related ones of the correct answer may be returned in the top-k recommendations which are actually correct. Taking these into consideration, the actual precision and MRR should be higher than the results presented in the figures.

5.3 RQ3: Temporal-Locality based Approach

Considering the temporal locality between the correlated issues and posts, in this section we test our approach described in Section3.4. We adopt VSM as the semantic similarity approach and use the fine-grained information *TTDA* to test for which achieve best results in the previous experiments.

As shown in Section 5.2, the results for recommending 100 candidates can achieve 67.38% and 83.94% for Android issues and SO posts respectively. This implies that for the majority of the issues/posts the correct answers are returned in the top 100 recommendations. Thus, in this experiment, we focus on optimizing the rank of the top 100 recommendations. We use temporal-locality to promote the ranks of these who arise temporally closer to the queries. The experiment results are presented in the Tables 7 and 6, in which *VSM* stands for the VSM-based semantic similarity approach, and *VSM+TL* represents the approach that integrates the semantic similarity with temporal locality. *TT* and *TTDA* represent the different granularities of texts used for calculating semantic similarity as defined in subsection 5.2.

The first row of the table shows VSM's performance based on tiles and tags, the second and third rows present the different approaches' performances based on fine-grained texts. As shown in the tables, the precision of top-5 based on *TT* is only 26.34% and 25.16% respectively. This is too poor to be applicable in practice. While taking more fine-grained texts and the temporal factors into account, the performance can be improved by about 20% and 30%. Comparing the approaches of considering temporal locality

TABLE 6
Performances for recommending posts to issues

Approach	Top 5		Top 10		Top 100	
	MP	MRR	MP	MRR	MP	MRR
VSM (TT)	0.2634	0.2143	0.2894	0.2178	0.4594	0.2236
VSM (TTDA)	0.3890	0.3027	0.4548	0.3110	0.6738	0.3187
VSM+TL (TTDA)	**0.4977**	**0.3832**	**0.5482**	**0.3899**	**0.6738**	**0.3954**

TABLE 7
Performances for recommending issues to posts

Approach	Top 5		Top 10		Top 100	
	MP	MRR	MP	MRR	MP	MRR
VSM (TT)	0.2516	0.1828	0.2985	0.1889	0.5089	0.1972
VSM (TTDA)	0.5300	**0.4156**	0.6111	**0.4267**	0.8394	**0.4355**
VSM+TL (TTDA)	**0.5700**	0.3822	**0.6683**	0.0.3968	**0.8394**	0.4035

or not, we can see that the temporal-locality can improve the precision. When recommending SO posts to Android issues, the precision for recommending 5 increases from 38.99% to 49.77% which is improved almost by 11.00%. While for SO post queries, the precision increases from 53.00% to 57.00%, which is improved by 4.00%. This proves the effectiveness of temporal-locality for improving recommendation accuracy, and our approach can capture such locality and take advantage of it.

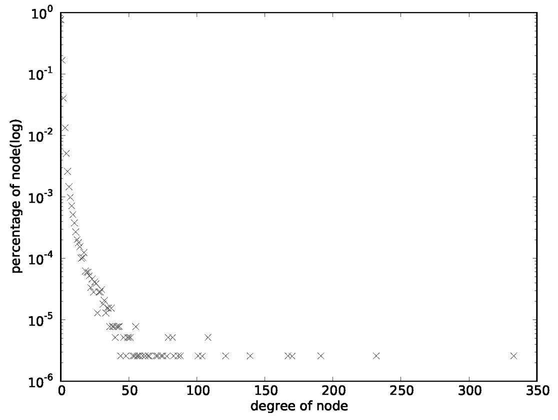
For MRR when recommending 5 candidates, it increases from 0.30 to 0.38 in querying Android issues while decreases from 0.41 to 0.38 in querying SO posts. The decrease for querying SO posts suggests that the temporal locality can promote some of the standard answers' ranks from after 5 to top 5, meanwhile it decreases the ranks of a few correct answers which are already in top 5. However, the overall decrease is very subtle. Besides, considering that we only focus on top 5 or top 10 results, the precision in our scenario is more important than MRR.

5.4 Citation-Graph based Approach

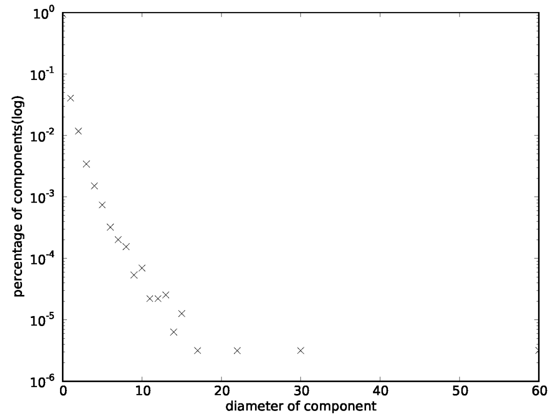
Because of the differences of the mechanisms and aims, recently the internal links in Android Issue Tracker are not so prevalent as that in Stack Overflow. Thus, for RQ4 we only do experiment on recommending SO posts to issues to validate the cluster-based approach. We first analyze the internal links in Stack Overflow and the semantic similarities among these connected posts. Then we group the posts into different closely coherent posts by using such information. In the end, we do linking recommendation based on post clusters instead of single post.

5.4.1 The characteristics of citation-graph in SO

We build the citation-graph based on the internal links in Stack Overflow. Figure 7 shows the characteristics of the citation-graph in Stack Overflow.



(a) The degree distribution of nodes in SO internal citation-graph



(b) The diameter distribution of connected components in SO internal citation-graph

Fig. 7. The characteristics of SO internal citation-graph

Figure 7(a) presents the log distribution of nodes degrees in citation-graph, in which the x axis stands for the degree and the y axis stands for the percentage of nodes with that degree. Overall, the degree of nodes follows a kind of long-trail distribution. A large proportion of nodes are isolated. With the increase of the degree, the number of nodes with that degree reduce constantly. In specific, there are more than 296 thousands nodes are of 0 degree, which take a proportion of 76.65% among all the nodes. While the node of post question with largest degree receives 31 answers and 91 comments, and the corresponding post thread is connected with 333 other post threads.

In the internal citation-graph, lots of nodes are connected through many other media nodes, and many connected components are formed. We analyze the connect components in the citation-graph, and Fig. 7(b) presents the diameter distribution of all the largest connected components. Similar to that in Fig. 7(a), Fig. 7(b) is a log distribution of component diameters. And the x axis represents the diameter, and the y axis represents the percentage of components with that diameter. In the citation-graph, we view isolated nodes as components with diameter of 0. Overall, the diameter distribution also follows a kind of long-trail distribution. In the graph, there are about 12,793 connected components with degree of 1. And the larger the diameter, the less the number of connected components. Among all the connected components, the diameter of the largest one reaches 61.

From Fig. 7 we can see that, among the nodes and connected components, some of the nodes are connected with quite a large number of other nodes, and many connected components are of large diameters. For such nodes and components, although there are links between them, they may not concentrate on the same core issue. Thus, we need to depart such nodes and components so as to find closely coherent nodes.

5.4.2 Experiment results based on citation-graph

We cluster the posts based on the semantic similarities and components diameters. By several iterations of experiments and manual analysis of clustered results, we set the similarity threshold as 0.3 and the diameter threshold as 4 which can cluster posts with best internal cohesion. Table 8 presents the changes of citation-graph before and after clustering.

We compare the number of connected components, the diameter and the number of nodes in the largest connected components before and after clustering in Tab. 8. We can see that the number of total nodes keep the same because we do not delete or add any nodes when clustering. But the number of edges reduces from 76,058 to 33,929. This implies that a lot of original edges are deleted because the semantic similarities between the connected pairs of nodes are low. While the number of connected components increases from 315,761 to 354,395. This is because a lot of original components are very large, and some nodes in the components are connected through many intermediate nodes. Such nodes in the large components are not so closely related, and thus be departed. This divides large components into many more small coherent components. As shown in the Table, the diameter of the largest component reduced from 60 to 4, and the number of nodes in it decreased dramatically from 34,003 to 82.

For a given query issue, we do linking recommendation based on such post clusters. We compare the performances of different approaches which based on the *S*emantic *S*imilarity(SS), combination of *S*emantic similarity and *T*emporal *S*imilarity(STS), and the combination of *S*emantic, *T*emporal based on citation *G*raph(STG) respectively for recommendation.

Figure 8 and Tab. 9 present the detailed results, which shows the precisions and MRRs for recommending different number of candidates ranging from 5 to 50.

As can be seen from the figure and the table, as the recommending number grows, the hit precision and the MRR will increase as well. Overall, STG approach performs

TABLE 8
The Statistics of citation-graph before and after clustering

	#nodes	#edges	#connected components	#isolated nodes	#nodes in largest component	diameter of largest component
Before	387,422	76,058	315,761	297,160	34,003	60
After	378,422	33,929,	354,395	337,766	82	4

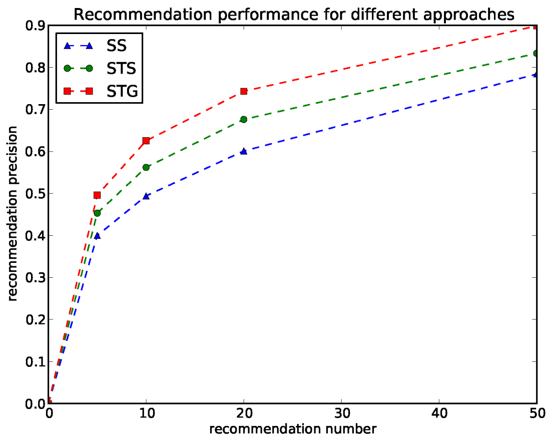


Fig. 8. Recommendation precision for different approaches

TABLE 9
MRR for linking Recommendation with different approaches

Approach	Top-5	Top-10	Top-20	Top-50
SS	0.2799	0.2926	0.2999	0.3057
STS	0.3226	0.3371	0.3451	0.3954
STG	**0.3530**	**0.3704**	**0.3786**	**0.3838**

best. When recommending top-10 candidates, STG approach reaches average precision of 62.51% and MRR of 0.3704. Compared with the previous work [11], we can see that our approach is significantly higher than the results presented in the paper which is only 33.16%. When recommending top-50, the precision reaches 89.85%. This means that, for about 90% of the testing issues, the truly correlated posts are corrected returned in the top 50 candidates. Comparing the three approaches, we can see that STG always performs better that the other two approaches no matter how many candidates are recommended.

One thing need to note is that, we only view these existing links in issues as the correct answers, and the others even though they are the duplicate of the standard answers will be viewed as incorrect. For some issues, although the standard answers are not contained in the top-k recommendations, the duplicate or related ones of the correct answer may be returned in the top-K recommendations which are actually correct. Although recommending clusters instead of single post will alleviate this problem, the actual precision should be higher than the results presented here.

6 Threats to Validity

There are some threats that may affect the effectiveness of our approach. In this paper we archive best performance when using the fine-grained texts of the discussion threads. For newly posted issues or questions the performance may decline. For this, as presented and discussed in section 5.2, the performance for recommendation based on title plus description can achieve acceptable results even only counting the standard answers, let alone there are lots of duplicates or closely-related issues and posts which may be recommend correctly but not viewed as correct answers.

Threats to external validity are mainly related to the generalizability of our approach. In this paper, we mainly focus on Android Issue Tracker and SO. In the future, we plan to conduct comprehensive experiments on more software communities including Eclipse, jQuery and others alike.

7 Related Work

The emerging software communities over the Internet appeal to all kinds of stakeholders, and the knowledge sharing and exchanging between these communities has attracted great interests from researchers.

Treude et al. [26] studied various web-based communication portals and channels. Their work proved the importance of effective knowledge management and sharing in software development. Gómez et al. [14] focus on SO and studied the knowledge sharing in it. They found that SO plays an import role in software innovation dissemination. Vasilescu et al. [27] compared the activities of shared authors who are active in both SO and r-help mailing list. This work suggested that the experts are migrating from mailing list to SO and provide faster answers in SO than they do in mailing list. These researches inspire and motivate us to study how to combine the issue tracking system and SO to automatic knowledge sharing between them and benefit each other.

To make full use of the knowledge dispersed in various communities, different approaches are proposed on linking those resources to assist various software development activities. Gottipati et al. design a semantic search engine framework to retrieve relevant posts from long thread discussions in software forums [28]. The authors firstly inference semantic tags for posts by manually constructing training dataset, and then utilize those tags to improve search accuracy. Brandt et al. [29] designed a web search interface and integrated it into IDE to help users locate example code from web pages when programming. Bacchelli et al. [10], [30] focus on utilize the crowd knowledge in SO to solve developers' programming-specific problems. They develop an Eclipse plugin to combine SO with IDE seamlessly, which can employ the programming context in IDE to formulate query and get most relevant discussion pieces. Rahman et al. [31] focus on finding working solution for runtime error or exceptions. They collect results from three general search engines and SO site, and then present the synthesized results in the IDE. These work focus on providing immediate help when programming/debugging in IDE, and heavily rely on code terms and the programming context to locate related resources. Different from them, our work aims at linking issues with SO discussions to help solving software issue and developers' questions, and the texts can be used are mainly written in natural language.

Besides, there are works on improving the mechanisms of traditional Issue Tracker System. To increase the completeness of bug reports, Zimmermann et al. [32] proposed four ways to improve current issue tracking system, including tool-centric, information-centric, process-centric and user-centric, and design an interactive system by asking reporters several questions to collect desired information from reporters and locate the reported bug. Different from this work, Lotufo et al. [25] investigate the game mechanisms in Stack Overflow, and study the applicability of these mechanisms to addressing the issues of current bug tracking system and motivate reporters contribute more frequently with high-quality. These researchers provides ways to improve current issue tracker system mechanically. Our work do not mean to change the traditional Issue tracking system, but resort to the online knowledge communities to acquire adequate feedback from users so as to provide more useful information of given bug and improve bug fixing efficiency.

To utilize the online forum knowledge for issue resolution, there has been a few attempts. Inozemtseva et al. [33] propose an approach to connect various project resources including API document, issue tracker, questions in SO and so on. They firstly extract structural names from source code, then by retrieving resources that have references to these names they connect the related resources. The limitation of the approach is that the resources must have structural name, or it will not be applicable. Correa et al. [11] explore the influence for including SO posts in issues, which is close to our work. They investigate the influence of links in Android and Chromium issues quantitatively and qualitatively. Their experiments and survey prove the effectiveness of including SO links in issue resolution. In addition, the authors propose approach to retrieve related SO questions by calculate title and tag similarity. However, the experiment results in the work show that only titles and tags are not enough for providing satisfying results. Different from this work which focus on unidirectional knowledge acquisition, we analyse the knowledge sharing from both directions and propose a uniform model which accomplishes this process efficiently and improve the performance significantly.

8 Conclusion and Future Work

In this paper, we use Android Issue Tracker and Stack Overflow as a case to study the automatic knowledge sharing across communities. We analyse the two different communities and the benefits of knowledge sharing between them. To mine potential associations between issues and posts in them to accomplish automatic knowledge sharing, we compare the capabilities of different retrieval approaches in this scenario and found that Vector Space Model performs best when using fine-grained information of issues and posts. We explore the temporal locality between the correlated issues and posts, and integrate it with semantic factors to build an uniform similarity model. Using the synthesized model, we can bridge the two communities by linking related issues to posts automatically. Our approach archives significant improvement in recommendation accuracy, with a precision of 54.82% at recommending 10 posts to issues and 66.83% in reverse. In addition, we explore the internal links in Stack Overflow to cluster posts. Based on this approach, the precision for recommending top-10 post clusters to issues reaches 62.51% which is improved by 7.69%.

Based on this motivation, we build the *OSSEAN* platform. It has crawled more than 330 thousands of projects and 9.6 millions of posts from several influential and prevalent collaborative development and knowledge sharing communities including GitHub, SourceForge, Stack Overflow, OsChina, SlashDot and so on. Currently *OSSEAN* can provide services like OSS ranking based on user feedbacks, discovering technology hot topics and so on. These services can be reached at *http://ossean.trustie.net* now.

In the future, we plan to apply our approach on more communities including bug tracker systems of Eclipse and jQuery and so on to validate the generality of our approach. Besides, the embedded gamification mechanisms in Stack Overflow are of great value for sifting high-quality knowledge, which will be studied and integrated to our approach to optimize the ranking of related recommendations further and provide more comprehensive discussions for issues. In addition, more OSS and posts will be analyzed, and other interesting and useful services will be provided in *OSSEAN* like technology trends analysis and so on.

Acknowledgments

This research is supported by the National Natural Science Foundation of China (Grant No.61432020 and No. 61472430), the National High Technology Research and Development Program of China (Grant No.2012AA011201).

References

[1] I. Rus and M. Lindvall, "Knowledge management in software engineering," *Software, IEEE*, vol. 19, no. 3, pp. 26–38, May 2002.

[2] J. Xu, Y. Gao, S. Christley, and G. Madey, "A topological analysis of the open souce software development community," in *System Sciences, 2005. HICSS'05. Proceedings of the 38th Annual Hawaii International Conference on*. IEEE, 2005, pp. 198a–198a.

[3] S. Diehl, H. C. Gall, and A. E. Hassan, "Guest editors introduction: special issue on mining software repositories," *Empirical Software Engineering*, vol. 14, no. 3, pp. 257–261, 2009.

[4] M. Allamanis and C. Sutton, "Why, when, and what: analyzing stack overflow questions by topic, type, and code," in *Proceedings of the Tenth International Workshop on Mining Software Repositories*. IEEE Press, 2013, pp. 53–56.

[5] C. Iacob and R. Harrison, "Retrieving and analyzing mobile apps feature requests from online reviews," in *Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on*. IEEE, 2013, pp. 41–44.

[6] L. Mamykina, B. Manoim, M. Mittal, G. Hripcsak, and B. Hartmann, "Design lessons from the fastest q&a site in the west," in *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 2011, pp. 2857–2866.

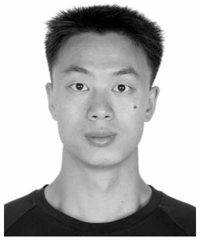
[7] A. Bacchelli, M. Lanza, and R. Robbes, "Linking e-mails and source code artifacts," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*. ACM, 2010, pp. 375–384.

[8] S. Subramanian, L. Inozemtseva, and R. Holmes, "Live api documentation," in *Software Engineering (ICSE), 2014 36th International Conference on*. ACM, 2014, p. To appear.

[9] N. Sawadsky, G. C. Murphy, and R. Jiresal, "Reverb: recommending code-related web pages," in *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013, pp. 812–821.

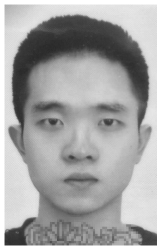
[10] A. Bacchelli, L. Ponzanelli, and M. Lanza, "Harnessing stack overflow for the ide," in *Recommendation Systems for Software Engineering (RSSE), 2012 Third International Workshop on*. IEEE, 2012, pp. 26–30.
[11] D. Correa and A. Sureka, "Integrating issue tracking systems with community-based question and answering websites," in *Software Engineering Conference, 2013 22nd Australian*. IEEE, 2013, pp. 88–96.
[12] A. Begel, J. Bosch, and M.-A. Storey, "Bridging software communities through social networking." *IEEE Software*, vol. 30, no. 1, 2013.
[13] M. Zhou and A. Mockus, "What make long term contributors: Willingness and opportunity in oss community," in *Proceedings of the 2012 International Conference on Software Engineering*. IEEE Press, 2012, pp. 518–528.
[14] C. Gómez, B. Cleary, and L. Singer, "A study of innovation diffusion through link sharing on stack overflow," in *Mining Software Repositories, 2013 10th IEEE Working Conference on*. IEEE, 2013, pp. 81–84.
[15] A. Barua, S. W. Thomas, and A. E. Hassan, "What are developers talking about? an analysis of topics and trends in stack overflow," *Empirical Software Engineering*, pp. 1–36, 2012.
[16] S. Rao and A. Kak, "Retrieval from software libraries for bug localization: a comparative study of generic and composite text models," in *Proceedings of the 8th Working Conference on Mining Software Repositories*. ACM, 2011, pp. 43–52.
[17] S. Kawaguchi, P. Garg, M. Matsushita, and K. Inoue, "Mudablue: An automatic categorization system for open source repositories," *Journal of Systems and Software*, vol. 79, no. 7, pp. 939–953, 2006.
[18] F. Thung, S. Wang, D. Lo, and J. Lawall, "Automatic recommendation of api methods from feature requests," in *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*. IEEE, 2013, pp. 290–300.
[19] T. Dasgupta, M. Grechanik, E. Moritz, B. Dit, and D. Poshyvanyk, "Enhancing software traceability by automatically expanding corpora with relevant documentation," in *Software Maintenance (ICSM), 2013 29th IEEE International Conference on*. IEEE, 2013, pp. 320–329.
[20] C. McMillan, M. Grechanik, and D. Poshyvanyk, "Detecting similar software applications," in *Software Engineering (ICSE), 2012 34th International Conference on*. IEEE, 2012, pp. 364–374.
[21] C. H. Papadimitriou, H. Tamaki, P. Raghavan, and S. Vempala, "Latent semantic indexing: A probabilistic analysis," in *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*. ACM, 1998, pp. 159–168.
[22] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *the Journal of machine Learning research*, vol. 3, pp. 993–1022, 2003.
[23] L. Singer, F. Figueira Filho, B. Cleary, C. Treude, M.-A. Storey, and K. Schneider, "Mutual assessment in the social programmer ecosystem: an empirical investigation of developer profile aggregators," in *Proceedings of the 2013 conference on Computer supported cooperative work*. ACM, 2013, pp. 103–116.
[24] E. Linstead and P. Baldi, "Mining the coherence of gnome bug reports with statistical topic models," in *Mining Software Repositories. 6th IEEE International Working Conference on*. IEEE, 2009, pp. 99–102.
[25] R. Lotufo, L. Passos, and K. Czarnecki, "Towards improving bug tracking systems with game mechanisms," in *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*. IEEE, 2012, pp. 2–11.
[26] C. Treude and M.-A. Storey, "Effective communication of software development knowledge through community portals," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. ACM, 2011, pp. 91–101.
[27] B. Vasilescu, A. Serebrenik, P. Devanbu, and V. Filkov, "How social q&a sites are changing knowledge sharing in open source software communities," in *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work*. ACM, 2014, pp. 342–354.
[28] S. Gottipati, D. Lo, and J. Jiang, "Finding relevant answers in software forums," in *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*. IEEE Computer Society, 2011, pp. 323–332.
[29] J. Brandt, M. Dontcheva, M. Weskamp, and S. R. Klemmer, "Example-centric programming: integrating web search into the development environment," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2010, pp. 513–522.
[30] L. Ponzanelli, A. Bacchelli, and M. Lanza, "Leveraging crowd knowledge for software comprehension and development," in *Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on*. IEEE, 2013, pp. 57–66.
[31] M. M. Rahman, S. Yeasmin, and C. K. Roy, "An ide-based context-aware meta search engine." IEEE, pp. 467–471.
[32] T. Zimmermann, R. Premraj, J. Sillito, and S. Breu, "Improving bug tracking systems," in *Software Engineering-Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on*. IEEE, 2009, pp. 247–250.
[33] L. Inozemtseva, S. Siddharth, and R. Holmes, "Integrating software project resources using source code identifiers," in *Proceedings of the International Conference on Software Engineering*, 2014.

Huaimin Wang received his Ph.D. in Computer Science from National University of Defense Technology (NUDT) in 1992. He is now a professor in department of educational affairs, NUDT. He has been awarded the "Chang Jiang Scholars Program" professor and the Distinct Young Scholar, etc. He has worked as the director of several grand research projects and has published more than 100 research papers in peer-reviewed international conferences and journals. His current research interests include middleware, software Agent, trustworthy computing. He is a member of the IEEE.



Tao Wang received both his B.S. and M.S. in Computer Science from National University of Defense Technology (NUDT) in 2007 and 2010. He is now a Ph.D. candidate in Computer Science in NUDT. His work interests include open source software engineering, machine learning, data mining and knowledge discovering in open source software.He is a student member of the IEEE.

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 and software engineering and machine learning. He is a member of the IEEE.



Cheng Yang received his M.S. in Computer Science from National University of Defense Technology (NUDT) in 2013. He is now a Ph.D. candidate in Computer Science in NUDT. His work interests include open source software engineering, machine learning, data mining and knowledge discovering in open source software.