G-Finder: Routing Programming Questions Closer to the Experts *

Wei Li [1,2,3] Charles Zhang [2] Songlin Hu [1]

[1] Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
[2] Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong, China
[3] Graduate University of Chinese Academy of Sciences, Beijing, China
liwei01@ict.ac.cn charlesz@cse.ust.hk husonglin@ict.ac.cn

Abstract

Programming forums are becoming the primary tools for programmers to find answers for their programming problems. Our empirical study of popular programming forums shows that the forum users experience long waiting period for answers and a small number of experts are often overloaded with questions. To improve the usage experience, we have designed and implemented `G-Finder`, both an algorithm and a tool that makes intelligent routing decisions as to which participant is the expert for answering a particular programming question. Our main approach is to leverage the source code information of the software systems that forums are dedicated to, and discover latent relationships between forums users. Our algorithms construct the concept networks and the user networks from the program source and the forum data. We use programming questions to dynamically integrate these two networks and present an adaptive ranking of the potential experts. Our evaluation of `G-Finder`, using the data from three large programming forums, takes a retrospective view to check if `G-Finder` can correctly predict the experts who provided answers to programming questions. The evaluation results show that `G-Finder` improves the prediction precision by 25% to 74%, compared to related approaches.

Categories and Subject Descriptors D.2.6 [*Software Engineering*]: Programming Environments

General Terms Design, Human Factors

* In programming communities, experts are synonymous to both Gurus and Geeks.

OOPSLA/SPLASH'10, October 17–21, 2010, Reno/Tahoe, Nevada, USA.

Keywords Expert Search, Social Networks, Programming Forums

1. Introduction

Programming forums are becoming a major means for programmers to get help for their programming tasks, especially when open source software packages developed by the community effort are involved. The basic unit of programming forums is a thread, consisting of a chain of messages, starting with a question message from a user (questioner) and a number of reply messages from other users, as illustrated in Figure 1. If a message $m1$ is to reply another message $m2$, $m2$ can be considered as the parent of $m1$. A message has one parent at most in a thread.

Programming forums are being actively used. The *Java Develop Forum*[1], an online system from *Sun*, contains 87 sub-forums that focus on various topics concerning Java programming [17]. Just one of these sub-forums, *Java Programming*[2] (referred to as the *Java Forum* hereon), contains about 77000 threads or 490000 messages. In this forum, there are over 200 participants online and 200 messages posted per day. The *Eclipse Forum*[3], a platform for users to discuss programming problems using the *Eclipse* platform, contains about 80 sub-forums, and some of these sub-forums contain more than 20000 threads. All these forums have thousands of users registered, and they interact with each other by posting messages and waiting for replies from other users.

Despite the popularity of programming forums, the forum participants often experience a few common problems. First, the time span between raising the initial question and getting the satisfactory answer is often hard to predict and longer than expected. Through our investigation of the *Java Forum*, we found that the time taken for a question to be

[1] Java Develop Forum. URL: `http://forums.sun.com/index.jspa`

[2] Java Programming Forum. URL: `http://forums.sun.com/forum.jspa?forumID=31`

[3] Eclipse Forum. URL: `http://www.eclipse.org/forums/index.php?t=index&cat=1&`

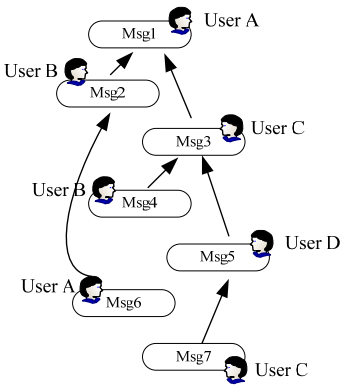

Figure 1: An Example of Thread
It contains seven messages and four users. The arrows show the post-reply relation

answered can range from less than an hour to 3 or 4 days or even longer. A question lingering for several days will undoubtedly slow down the programming productivity. Second, experts in forums are overloaded with questions, which affects the quality and timeliness of the answers. We randomly sampled about 500 questions and 600 participants from the *Java Forum*. We found that only one third of participants give replies, and 14% of these participants have given 61% of the replies. All of the observations we made above call for an intelligent information processing tool that can make good decisions with respect to who is the right person to answer a programming question.

The problem of expert searching has been extensively explored in the information retrieval research. We classify the previous work using the simple criteria of whether they use the content of the messages or the post-reply relationships between participants or both types of knowledge in the analysis. The language model proposed in [2][3] computes the probabilities of the term appearance in plain texts of the participants. Another line of work [6][17] computes the global rankings of the forum participants through the construction of the participant graphs and the use of well-known algorithms such as the PageRank [5] and HITS [7]. The constructed participant graphs encode the post-reply relationships of participants in the message threads. The content of the messages is, however, ignored. A recent method [18] combines the language model and the post-reply structure to improve the precision and recall of the expert searching in forums. It uses term probabilities to profile the forum participants. These profiles are combined with a global ranking approach computing from the post-reply graphs to produce the final ranking of experts. This combination is reported to have improved both the precision and the recall of the expert recommendation in forum systems.

Compared to the general forums that the related approaches are concerned with, programming forums (PFs) share a certain degree of similarity and, at the same time, exhibit some important differences. One difference is that the messages in PFs contain not only texts but also, in most cases, fragments of code. It is problematic for the term probability calculation because it treats every word non-discriminatively. For instance, a forum question asking about how to use `I/O` operations in Java may present a piece of code fragment containing the `Java I/O` class: `BufferedReader`. The word, `BufferedReader`, despite having the same appearance weight as many other words in the message, is more important than others in representing the nature of the question and much more helpful in finding the right expert. Another difference is regarding the topics of the threads. In previous approaches [17] [18], different threads are considered representing independent topics. But for PFs, many threads have implicit relationships among each other. For example, experts participating a discussion thread involving the type, `java.io.BufferedInputStream`, are very likely capable of contributing to another thread involving `java.io.FilteredInputStream`, a super type of `BufferedInputStream`. Considering them as independent threads may lead to imprecision in the identification of experts.

Leveraging these observations, we have designed, implemented, and evaluated `G-Finder`, both an algorithm and a tool that uses the semantic information of forum threads, in addition to the post-reply relationships, to locate experts in programming forums. We encode the relationships of concepts and participants as graphs or "networks". Meanwhile, we associate forum threads and querying questions with a set of Java class types, we hereon referred to as *concepts*, using a few effective heuristics based on the observed patterns. We then construct the concept networks by extracting the relationships of concepts from either the source code or the bytecode of the system that the forum is dedicated to. We also build the user networks based on the post-reply relationships. However, unlike the related work, our user network is not static but computed on demand according to the concept-mappings of the queries. Therefore, our prediction is quite adaptive to the semantics of the queries and, hence, more accurate.

The evaluation of our algorithm is based on the real forum data crawled from three popular online programming forums. From these forums, we first extract "predictable" data by selecting questions that are answered by participants who also answered multiple questions. We then randomly divide the data into two sets, where the first set is used to construct both the concept and the user networks, and the second to test if `G-Finder` can correctly identify the experts who also answered the questions in the first set. We sampled about three to four thousand threads from two general *Java* programming forums and one specific programming forum from *Eclipse*, and evaluated the precision of our algorithm. Our evaluation shows that, compared to the previous work, `G-Finder` improves the prediction accuracy by 25% to 74%.

The main contributions of our paper are:

1. We present an empirical study of the post-reply relationships on three popular programming forums to further motivate the need for the more accurate and balanced selection of experts, for answering questions in online forums.
2. We present the design and the implementation of an algorithm that can be used by programming forums to route programming questions closer to the experts. This algorithm uses the semantic information extracted from the source code as the main insight for improving the prediction accuracy.
3. We evaluate the prediction quality of our approaches using the data from active programming forums. We also quantify our improvement with respect to the state of the art by implementing and comparing to the other related expert searching algorithms.

The rest of the paper is organized as follows: Section 2 shows the related research on expert searching; Section 3 presents some analysis that motivates our design principles. Section 4 shows our algorithm of expert searching on programming forums. Section 5 describes the implementation of our model. Section 6 shows the evaluation of our algorithms and discusses some observations on programming forums. Section 7 gives the conclusion and future work.

2. Related Work

The research of expert searching is originally based on the scientific and enterprise data. The research in [9] studied a topic model that matches papers with reviewers, which uses the method of Latent Dirichlet Allocation (LDA) [4] to obtain the distribution of documents over topics and the distribution of topics over the vocabulary of topics. Our method is different in following aspects. First, the topic model describes topics by the most probable words in the paper corpus, whereas, in PFs, the concepts are described by words concerning with the source code. Second, LDA is not suitable for PFs, as PFs have explicit word vocabularies of concepts. Third, the post-reply structures do not exist in the paper corpus. The significant research effort[4] is devoted to expert searching based on the enterprise data that reflects the experiences of users in real organizations, such as intranet pages, email archives, and document repositories, to identify experts. For this type of data, language models are the dominating techniques [2][3][11], which calculate the probabilities of term generation of the users through the documents associated with the users. The probability that a user will be an expert on a question is calculated through computing the probability that the user generates the set of terms in the question. Language models have sound foundations in theory [12][16] and perform well on the document corpus. Our algorithm is different from those that use language model in two aspects. First, our algorithm analyzes both the message content and the code structures of the target software system, hence, uses more information to improve the precision of the prediction. Second, the document corpus have no post-reply structures.

Regarding the expert search using forum data, Zhang et al [17] analyzed the characteristics of post-reply structures of the *Java Forum*, and compared some ranking methods including ranking using the PageRank values and the HITS authority values. Another method proposed by Jurczyk et al [6] constructs the ask-reply structures in Question/Answering portals, such as Yahoo!Answers[5], and uses link analysis methods, such as HITS, to find the users with high authority values or users answering a large proportion of questions. These methods ignore the content of the posts and rank users globally. Consequently, questions will always be routed to top ranked users, which exacerbates the overloading problem. In addition, global ranks also do not recognize the semantic differences of the questions and, consequently, result in low precision in identifying experts.

Zhou et al [18] proposed three models to route questions to proper users, including the profile-based model, the thread-based model, and the cluster-based model. These methods build language models with three different policies, which are integrated with a global ranking of users using the authority values, computed by the HITS-based algorithm on the post-reply graph. Their algorithms made significant improvement on precision and recall on general forums. Compared to this approach, we leverage the source code information of the target software system of the forum.

There is also significant research effort in the expertise recommendation in the software engineering area [10][1]. Mockus et al used the experience atoms [10] to identify the expertise. John Anvik et al [1] proposed methods on deciding who should fix bugs based on text classification methods. `G-Finder` aims to search experts in PFs, which is complimentary to these approaches.

3. An Empirical Study of Program Forums

To further motivate our research, we have conducted an empirical study on the post-reply structures of online programming forums. A special web crawler is implemented to crawl three popular forums, including the *Java Forum*, hosted by *Sun*[6] for developers to ask general JDK related questions, the *Java DevShed Forum*[7], another *Java* forum for general *Java* programming problems with about 22000 threads, and the

[4] Enterprise Track of TREC. URL: `http://trec.nist.gov/data/enterprise.html`

[5] Yahoo Answer. URL: `http://answers.yahoo.com/`

[6] Java Forum. URL: `http://forums.sun.com/forum.jspa?forumID=31`

[7] Java DevShed Forum. URL: `http://forums.devshed.com/java-help-9/`

GEF Forum[8], where the *Eclipse*[9] users search for help with the programming tasks using *GEF*. The size of the crawled data is about 2GB, consisting of 23000 threads from the *Java Forum*, 21000 threads from the *Java DevShed Forum*, and 7000 threads from the *GEF Forum*. As summarized in Table 2, we analyzed about 4000 threads from two *Java* forums and 3000 threads from the *GEF Forum*. We also collect the number of users as well as the number of distinctive concepts as the measures of the discussion diversity. In Table 2, we report the number of active users for each PF. The active users are defined as those giving more than 10 replies in our data set.

We formulate our analysis of the crawled forum data as the following observations:

Observation 1: The life span of discussions is long. Table 1 shows the distribution of the life span of the threads on the three investigated forums, where the life span is defined as the time period from the first message to the last message. It shows that about 43% of the threads last for more than 4 hours and about 21% of them last for more than one day on average. A significant number of discussions in the forums last for a long time, while the programmers want to solve their problems as soon as possible.

Forum	*0 - 4h*	*4 - 24h*	*24 - 48h*	*48h -*
Java Forum	52.4%	23.1%	10.3%	14.2%
Java Devshed	53%	20.3%	17%	9.7%
GEF	65.7%	20%	4.1%	10.2%

Table 1: Overview of PFs

Observation 2: Most of the questions are answered by a small number of active users. Table 2 reports the number of concepts, the number of users, and the number of active users in the three forums. It shows that the active users only constitute about 13% of all of the users. In Figure 2, we show the distribution of the number of users with respect to the number of replies given by a single user. Each point in the graph represents the distribution of the total number of users who provided more than the number of replies represented by the X-axis. For all of the three PFs, we observed that the measured overall "activeness" of users in providing answers and participants in discussions is distributed very unevenly, as most of the users provide a small number of replies, and the active users carry the majority of the workload. This type of imbalance detriments the usage experience of these forums. We believe that this power-law distribution is not a measure of the "willingness" of users to provide answers. Routing questions to proper users can not only lower the loads of active users, but also increase the participation of otherwise less active users, with the help of some incentive mechanisms, such as the money rewards in the *Amazon's Mechanical Turk Scheme*[10].

Forum	*Thread* Num	*Concepts*	*Users*	*Active* User
Java Forum	4000	814	3439	502
Java Devshed	4000	753	1549	186
GEF	3000	43	1373	110

Table 2: Overview of PFs

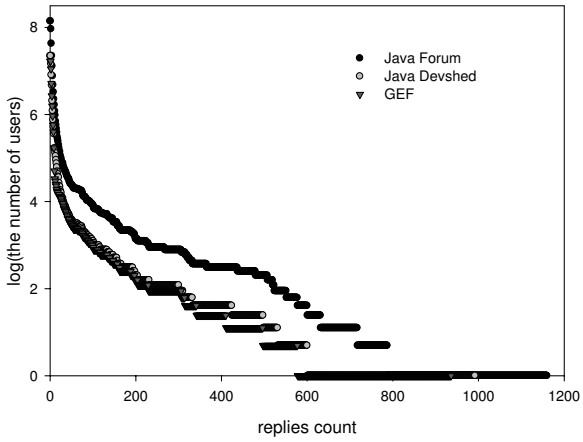

Figure 2: User/Replies Count Distribution

Observation 3: Experts answer questions in semantic clusters. We found that experts who answer multiple questions tend to answer questions that are related to each other. We analyze the relationships of concepts that participants give replies to on the *Java Forum*. It is not surprising that the replies of some users involve more than one class types. In fact, about 68% of the users participate in the questions related to at least 2 class types. What is more interesting is that about 75% of these participants give replies on the class types that have relationships. For example, in Table 3, a user called "jverd" in the *Java Forum* replies to questions on `HashMap` and `Map`, where the former is a subclass of the latter. `JComboBox` and `JTable` of the user "PhHein" are inherited from the same class type. The fact that one class type inherits from another, or two class types inherit from a third class type, which we refer to as the type hierarchy relationship, is common in the set of classes that each user answers. The relationships that span more than one level of inheritance is rare in our observation. Another implicit relationship among the class types is the call graph relationship. The fact that experts answer questions in semantic clusters gives us the inspiration to construct a relation network of the class types discussed by the threads, which is used to improve the quality of expert recommendation.

[8] GEF is an open source framework for GUI programming. URL: `http://www.eclipse.org/forums/index.php?t=thread&frm_id=81&`

[9] Eclipse Platform. URL: `http://www.eclipse.org/`

[10] Amazon's Mechanical Turk. URL: `https://www.mturk.com/mturk/welcome`

Table 3: Distribution on Concepts of Users

UserId	*Concepts*
DarrylBurke	`JList,JFrame,JButton,JScrollBar`
JoachimSauer	`String,Integer,List,ArrayList, Collection, HashSet, Collections`
jverd	`String,Integer,Map,Short, Boolean, ArrayList, HashMap`
DeltaGeek	`Array,Iterator,ArrayList, Collection`
PhHein	`JComboBox,JTable`
DrClap	`String,JTextField,List,Iterator, ArrayList`

From the observation 1 and 2 above, we believe that for the programming forums, an expert searching tool is useful in shortening the time span for the questions to be answered. Meanwhile, this tool should be capable of locating experts adaptively with respect to the semantics of the questions to release the load of experts in general. Observation 3 shows that some users tend to be "local experts", meaning that users tend to only answer questions on related concepts. For a specific concept, finding out potential experts on related concepts may help to retrieve more experts due to the clustering phenomenon. Driven by this objective, we have developed an adaptive expert searching algorithm that simultaneously considers message semantics and the social networks of users. We explain our algorithm in detail next.

4. Algorithm

The goal of the expert recommendation is to identify a list of participants who are knowledgeable about a given question. In `G-Finder`, this is achieved in the following steps, shown in Figure 3. First, we crawl down the pages of threads from the online forums. Second, we extract the concepts that each thread discusses. The concepts of a thread represent the main semantic content of the thread. Third, we build the concept networks to present the relationships among concepts, and the user networks to describe the post-reply structures of users. To search experts, we map the queries onto concepts and return a ranked list of users as recommended experts for each query.

Our algorithm is underpinned by a probabilistic model that calculates the probability of a user being an expert on a specific question. Given a question(or a thread), the probability of a participant being an expert on this question can be represented as the conditional probability $P(user|question)$. $P(user|question)$ can be calculated by multiplying the probability of user being an expert of a particular concept, $P(user|concept)$, with the probability of question belonging to the concept, $P(concept|question)$, summing over all the concepts:

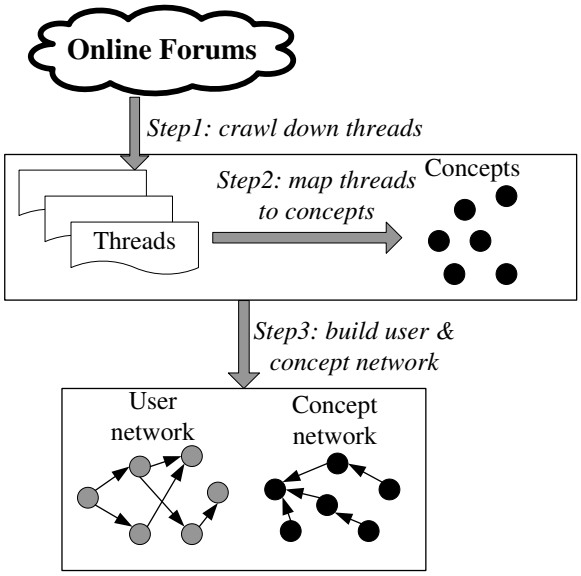

Figure 3: Algorithm Steps

$$P(user|question) = \sum_{concept} P(user|concept) \times P(concept|question) \quad (1)$$

$P(concept|question)$ is calculated using some heuristic methods, and $P(user|concept)$ is computed by constructing the user network on each concept, and integrating the user network with concept network.

4.1 Map Threads to Concepts

Concepts are the semantic foci of a thread. In PFs, many threads contain the class type information, and we believe that these class types, either referred to in the titles as well as the texts of messages, or embedded in the source code, can be considered as the main semantic narrative of the thread. We define the concept universe of a program forum as all the class types in the source code of the target software system. For example, for a *Java* forum, the concepts are referred to as the class names in JDK packages, and, in *GEF*, the concepts are the classes in the source code of the GEF packages.

We use a set of heuristics to calculate which of the concepts in the concept universe are relevant to a particular thread. The probability of a question belonging to a concept, $P(concept|question)$, is computed using the following heuristics:

1. **Title:** If the title of a thread contains n concepts, we define:
$$P(concept|question) = 1/n \quad (2)$$
Usually, the title of a thread is a short description of the question. The class types appearing in the title are strong indicators of what the question is concerned with.

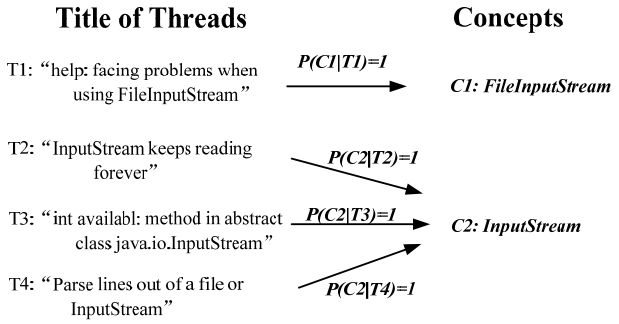

Figure 4: Map from Thread to Concept

Equation 2 states that a thread is represented by all the concepts in the title with equal probabilities.

2. **Source Code:** If the thread contains fragments of source code, we use a method similar to the calculation of the `tf/idf` [13] value. The `tf/idf` value shows the strength of the relevance of a term with respect to a document. `Tf` is the frequency of a term appearing in this document, and `idf` is the inverse of the frequency a term appears in all the documents. The `tf/idf` value of a term in a document is proportional to the relevance between the term and the document. In our context, we just consider the terms involving concepts and calculate the `tf/idf` values for each concept. We define `tf` as the frequency of a concept appearing in a thread among all the concepts in the thread and `idf` the inverse of frequency that a concept appears in all threads. Let $v_{concept\_question}$ be the `tf/idf` value of a concept in a thread, which indicates the relevance between the concept and the thread. If $v_{concept\_question}$ is higher, the thread is more likely concerned with the concept. We define $P(concept|question)$ as:

$$P(concept|question) = \frac{v_{concept\_question}}{\sum_{concept} v_{concept\_question}} \quad (3)$$

3. **Content:** In some threads, concepts are mentioned not in the source code but in the plain text of the messages. In this case, we consider the probability of the thread belonging to the concept, $P(concept|question)$, will be also $1/n$, where n is the number of concepts appearing in the content of the thread.

If a thread satisfies more than two criteria above, we just use one heuristic following the priority order of title, source code, and content. Figure 4 shows a simple example, in which four threads on the left have concepts in their titles, and are mapped to corresponding concepts. The values of $P(concept|question)$ are presented above the arrows in Figure 4. This example will be used through out this section to illustrate the complete calculation.

4.2 Probabilities of Participants on Concepts

The probability $P(user|concept)$ is calculated through two steps. We first establish the user network for each concept and compute the probability of a particular user being an expert on the concept, $P_{sep}(user|concept)$, based on the user network of the concept. In this step, the relationships of concepts are not involved. In the second step, we take the relationships among concepts, represented by concept network, into consideration to compute $P(user|concept)$ of a user, based on the semantic clustering phenomenon of concepts. Experiments show that this step can significantly improve the quality of expert recommendation.

4.2.1 User network

In forums, the post-reply structures of messages can be viewed as probabilistic voting networks for experts. For example, if user A replies to user B, user B is likely to vote user A as an expert. If multiple users reply, user B is equally likely to vote any of the repliers as an expert. If the user A replies multiple times, this will increase the probability of being voted as an expert. We extract this post-reply information from each thread and construct the user network $G(V, E)$ as follows. V is a set of nodes representing participants. A directed edge e from A to B signifies that user B has provided replies to user A. Each edge has a weight label, w_{AB}, which is the count of the number of replies.

After constructing the user network for each thread, we apply a consolidation step across all networks as follows. We merge the common nodes, i.e., the node representing the same participants, of the networks, and connect all the edges to the new node. If the edges are common, we do not add new edges and instead, simply update the weight of the common edge to be the sum of the weights of all the merged edges. This merging process is depicted in Figure 5, where Figure 5(a) shows the thread-specific user networks in the example of Figure 4 and Figure 5(b) shows the merged network.

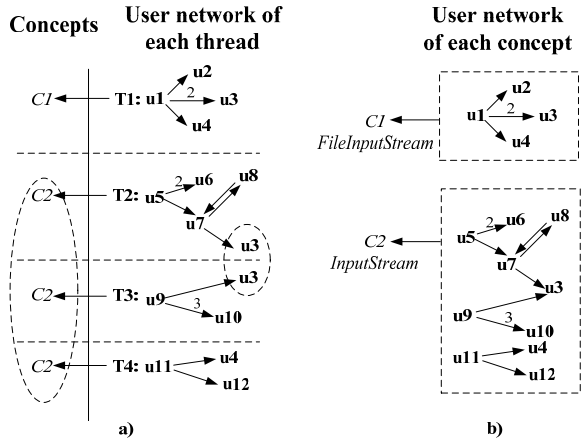

Figure 5: User Network
The concepts and the users in ellipses are merged. The numbers above the arrows are the counts of replies.

Based on the merged user network, we can compute the ranking of expert participants based on the voting scheme. Assume that the expertise value of a user u on a particular concept is $P_{sep}(u|concept)$. The probability of u voting v as an expert is P_{uv}, the fraction of the edge weight over the sum of weights of all the outgoing edges of u. Therefore, the probability of participant v being an expert on the same concept is $P_{uv} * P_{sep}(u|concept)$. Considering all the connected participants in the network, the complete calculation of $P_{sep}(v|concept)$ is as follows:

$$P_{sep}(v|concept) = \sum_{u}(P_{uv} * P_{sep}(u|concept)) \quad (4)$$

Let EV be the vector recording the likelihood probability for all users being an expert on a particular concept, and M be the voting probabilities matrix between all pairs of participants, Equation 4 encodes the voting calculations of the user network, which is a Markov process.

$$EV = M^T * EV \quad (5)$$

With some standard manipulations, we can use the PageRank algorithm [5] to compute $P_{sep}(u|concept)$ for each user. The column of $P_{sep}(u|concept)$ in Table 4 shows the probability values of users for the example in Figure 5.

Table 4: User Probabilities

Users: (***u1***: *luck2000@gmx.at* ***u2***: *JoachimSauer* ***u3***: *ejp* ***u4***: *thomas.behr* ***u5***: *kazenofairy* ***u6***: *JosAH* ***u7***: *gogo_* ***u8***: *pm_kirkham* ***u9***: *enfiend* ***u10***: *sabre150* ***u11***: *kimos2* ***u12***: *pieblok*),
Concepts: (***c1***: *FileInputStream* ***c2***: *InputStream*)

$P_{sep}(u\|concept)$	
$P_{sep}(u1\|c1) = 0.069$	$P_{sep}(u6\|c2) = 0.072$
$P_{sep}(u2\|c1) = 0.088$	$P_{sep}(u7\|c2) = 0.088$
$P_{sep}(u3\|c1) = 0.088$	$P_{sep}(u8\|c2) = 0.074$
$P_{sep}(u4\|c1) = 0.088$	$P_{sep}(u9\|c2) = 0.051$
$P_{sep}(u3\|c2) = 0.087$	$P_{sep}(u10\|c2) = 0.064$
$P_{sep}(u4\|c2) = 0.064$	$P_{sep}(u11\|c2) = 0.064$
$P_{sep}(u5\|c2) = 0.050$	$P_{sep}(u12\|c2) = 0.064$

In the previous work [17], the post-reply structure is constructed globally and used to compute PageRank or authority values as users' prior probabilities of being experts. On the contrast, our user network is built separately for each main concept of the queries. As a result, our user network is adaptive to queries, as the concept mappings of the different queries will vary. In the next step, we consider the relationships among the concepts and integrate the user networks adaptively according to the specific queries.

4.2.2 Concept Network

As we mentioned, participants tend to answer questions of related concepts. We consider two types of relationships in this paper and use them to construct the concept network.

1. **Type Hierarchy:** The first type of relationship is the type hierarchy. We consider two classes are related by the type hierarchy if one inherits another or both inherit the same super type. However, the transitivity of type hierarchy is not considered, as in Observation 3, we find the transitivity of type hierarchy rarely observed in related concepts. Given a class type A, the type hierarchy relation we consider is just the super classes, the children and the siblings of A in type hierarchy tree.
2. **Call Graph:** The second type of relationship we consider is the call graph relation. If a class A calls some methods of a class B in the source code, these two classes have the call graph relationship. We do not consider the entire call graph of the whole concept universe, only the concepts involved in the threads, generated by the code analysis tools[11].

After defining the relationships between concepts, we further define the weights of these relationships, to show the likelihood probability of a participant being an expert on both the related concepts. The weight of type hierarchy relationship, $w_h(a, b, u)$, and the weight of call graph relationship, $w_c(a, b, u)$, are defined as the likelihood probabilities of the user u being an expert on concept B, when the user is also an expert on concept A. These likelihood probabilities are determined by the specific data sets of forums. For example, if a user replies to three concepts that have the type hierarchy relationships with concept A, and there are totally ten concepts that have type hierarchy relationships with A, we consider the probability of this user to answer the concepts related to A to be $3/10$. We formally define the weights as follows:

$$w_h(a, b, u) = \frac{\mid concepts_{u\_reply\_to} \mid}{\mid concepts_{typeHierarchy-with-A} \mid} \quad (6)$$

And $w_c(a, b, u)$ can also be calculated similarly, with the relation type being the call graph relation:

$$w_c(a, b, u) = \frac{\mid concepts_{u\_reply\_to} \mid}{\mid concepts_{callGraph-with-A} \mid} \quad (7)$$

With the weights of these two relationships, the total weight between concept A and B of the user u is the sum of these two weights when the relationships exist. Define $h_{ab} = 1$ if type hierarchy relation exists and $h_{ab} = 0$ else from A to B, and define $c_{ab} = 1$ if call graph relation exists and $c_{ab} = 0$ else from A to B, the weight $w(a, b, u)$ from A to B of u is:

$$w(a, b, u) = w_h(a, b, u) * h_{ab} + w_c(a, b, u) * c_{ab} \quad (8)$$

The ***Concept Network*** is defined as a graph of concepts: $CN = \{V, E, P\}$, while the vertex set V represents the set

[11] In this research, we use the tool Wala. URL: `http://wala.sourceforge.net/wiki/index.php/Main_Page`

of concepts. P is the set of likelihood probabilities $P(a, b, u)$ for the user to be an expert on concept B when also being an expert on concept A, which is defined as follows:

$$P(a,b,u) = \begin{cases} w(a,b,u), & w(a,b,u) \leq 1, \quad (9) \\ 1, & w(a,b,u) > 1. \quad (9') \end{cases}$$

The likelihood probability $P(a, b, u)$ uses the two heuristics above and, when the sum of them is big enough, we can conclude that the probability is high enough to be defined as 1. E is the edge set and for a user, u, an edge from vertex a to b exists, when $P(a, b, u) > 0$. Note that for different users, the concept networks are also different as the weights may vary. Taking $u3$ in Figure 5 as an example, the left area of Figure 6 shows the concept network of $u3$, if the likelihood probabilities for $u3$ between $c1$ and $c2$ are both 0.5.

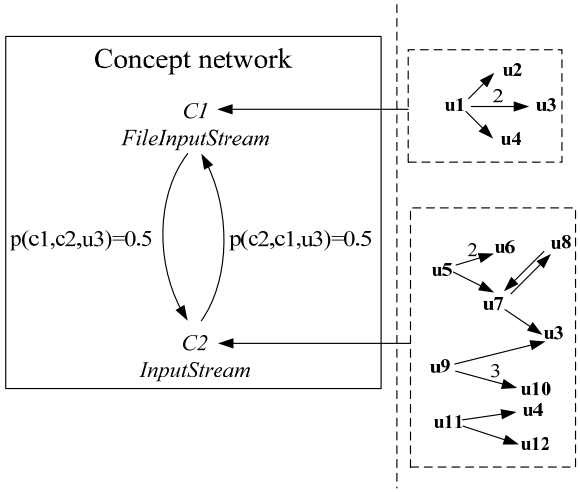

Figure 6: Concept Network

4.2.3 Integration of User Network and Concept Network

The concept network defines that a participant u, an expert on concept A with the probability $P_{sep}(u|concept_a)$, is also likely an expert on a related concept B with the likelihood probability of $P(a, b, u)$. So it can be considered that the expertise probability of the participant on concept B, as a result of being an expert on concept A, is $P(a, b, u) \times P_{sep}(u|concept_a)$.

Taking the relationships of concepts into consideration, we calculate the probability of a participant being an expert on a specific concept A, $P(user|concept_a)$, by accumulating all the $P_{sep}(u|concept)$ of u on related concepts of A, shown in concept networks. As the user tends to answer a related concept B with the likelihood probability, $P(b, a, u)$, we determine $P(u|concept_a)$ in following Equation:

$$P(u|concept_a) = \sum_b P(b,a,u) \times P_{sep}(u|concept_b) \quad (10)$$

When $a = b$, we consider $P(a, a, u) = 1$. The probability $P(u|concept_a)$ can be viewed as combining the probabilities, $P_{sep}(u|concept_b)$, of the same user being an expert on all of the concepts associated with A with the strength, $P(b, a, u)$, of the associations.

4.3 Query Processing

We describe how we leverage both concept networks and user networks to process queries. A query q is usually a post with source code. The query is first mapped to a set of concepts as described in Section 4.1, in which $P(concept|q)$ is calculated. The probability of $user$ being an expert on $concept$ is calculated by Equation 10. With Equation 1, the probability of the $user$ being an expert of the question q is calculated through following equation:

$$P(user|q) = \sum_{concept_a} (\sum_b P(b,a,user) P(user|concept_b)) \times P(concept_a|q) \quad (11)$$

For the example in Figure 5, assuming a query, q, the title of which is *"I have a question on how to use FileInputStream"*, is analyzed by `G-Finder`. First, q is mapped to only one concept, $c1$, with the probability of 1. According to Equation 11, the probability $P(u3|c1)$ is the sum of the probability of $u3$ being an expert on $c1$, and the probability of $u3$ on related concept $c2$ multiplied by the likelihood probability $P(c2, c1, u3)$. Therefore, the calculation is $0.088 + 0.087 \times 0.5 = 0.1315$. If another query is mapped to $c2$ with the probability of 1, $P(u3|c2)$ would be $0.087 + 0.088 \times 0.5 = 0.131$.

Our model produces adaptive results for each query, as each query has different probabilities $P(concept|q)$ on concepts according to their differences in the attached source code or their contents.

5. Implementation

We have implemented our algorithm in a tool called `G-Finder`, the architecture of which is shown in Figure 7. The wide arrows show the flow of data preprocessing: the forum data and the software source code that the forum is dedicated to are downloaded by the crawler, and the raw data preprocessor takes the raw forum data and outputs the threads into a relational database. These two components are described below:

1. **Crawler:** The crawler downloads the pages of threads from online forums, given the seed links of the forums. The seed links are the initial links that the crawler uses to download pages, of which the links are extracted for further downloading. As a forum always has fixed site structures and page formats, our crawler efficiently extracts new links of pages in the seed pages using regular expressions.

2. **Raw data preprocessor:** After downloading data, the raw data preprocessor extracts the posted messages, the users, the post-reply structures, and the source code from the pages. This information is then stored into database tables.

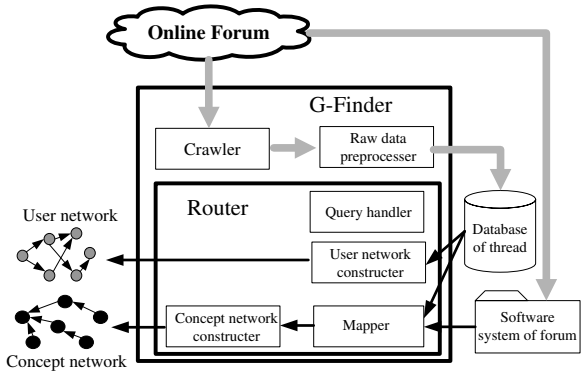

Figure 7: Architecture

The thin arrows show the steps of the model construction. The key module is the router, which extracts the threads from the database and constructs both the concept networks and the user networks. The router also processes queries and returns the ranked user lists. All these functions are achieved through four sub-components of the router described below:

1. **Mapper:** The mapper maps the threads to concepts, taking the threads in the database and the code of software system pertaining to the forum as input. The threads are mapped to concepts with certain probabilities using the heuristics explained in Section 4. The code of software system of forum is downloaded separately.
2. **Concept network constructor:** The concept network constructor builds the concept networks by constructing the relationships of the concepts and computing the weights among concepts using the threads in the database. The type hierarchy relationship and the call graph relationship between classes are obtained using the code analysis tool `Wala.`[12]
3. **User network constructor:** The user network constructor builds the user networks for each concept, and calculate the probability $P(user|concept)$ using the post-reply structures in the threads.
4. **Query handler:** The query handler, taking queries as input, calculates the probabilities $P(concept|question)$ by invoking the functionalities of the mapper, and then returns a list of ranked users.

[12] Wala. URL: `http://wala.sourceforge.net/wiki/index.php/Main_Page`

6. Experiment

Our evaluation of `G-Finder` aims at the following objectives:

- *Generality*. Does our algorithm effectively work on programming forums in general?
- *Effectiveness*. Is our technique more effective compared to related approaches?
- *Scalability* Does our modeling technique scale with the size of the forum data?

We crawl the data of discussion threads from three forums: *the Java Forum*, *the Java DevShed Forum*, and *the GEF Forum*. Our main evaluation method is to first create a manually verified historical data set as the oracle. We then implement an array of related approaches and compare the performance of `G-Finder` against them. In the rest of the section, we first describe the oracle data set, followed by the details of the evaluation.

6.1 Oracle Data Set

Unlike the earlier evaluation method [18], which is based on the manual judgement to assess the precision, we use the historical data to quantify the prediction effectiveness of our algorithm. We divide the concepts concerned in the threads into two parts randomly, one called `PA`, used to construct both the user and concept networks, and the other called `PB`, used to test the results. After division, we only reserve the "predictable" data in these two sets, such that the threads in one set have participants who also provided answers to the questions in the other set. We compute the probabilities that the user can answer the associated questions in `PB`, according to the model built using `PA`. The returned user list is compared with the actual expert set, defined as $AU_{concept}$.

The expert set, $AU_{concept}$, is obtained through a consensus voting process on each thread. Six graduate students were involved to go through the questions in `PB` and vote which user is the expert. We pick the users with more than 4 votes as the experts and add them into $AU_{concept}$. Then the precision of our algorithm is computed by comparing the returned user list with $AU_{concept}$. The manual judgement of experts can not be avoided for the forum systems that do not tag who have given the right answers to questions. However, the voting approach downplays the subjectivity in our experiments.

6.2 Metrics

In the expert searching area, the metrics of mean of average precision (MAP) [14][15] and Precision@N (P@N) [8] are widely used to evaluate the precision of expert searching. They are defined as follows:

DEFINITION 1. ***MAP:*** *MAP is the mean of the average of precisions over a set of query questions. The average of precisions for a query is the average of precision at each correctly retrieved answer(expert).*

Table 5: Performance on Three Java Forums

Method	*Java Forum*			*Java Devshed*			*GEF*		
	MAP	P@5	P@10	MAP	P@5	P@10	MAP	P@5	P@10
G-Finder	0.73	0.65	0.73	0.74	0.65	0.72	0.67	0.61	0.63
Profile-based	0.58	0.55	0.53	0.61	0.57	0.62	0.57	0.54	0.55
Language Model	0.53	0.53	0.54	0.55	0.56	0.57	0.52	0.51	0.53
Replies Count	0.43	0.41	0.4	0.51	0.51	0.52	0.33	0.31	0.34

DEFINITION 2. ***P@N*** *is the percentage of top* ***N*** *candidate answers who are correct.*

6.3 Benchmarked Approaches

We compare our model to three methods, including the language model [2], the profile-based method [18], and the method of ranking using the reply count [18]. The detailed descriptions on these three methods are below:

1. **language model:** The language model calculates the probability of a user to generate the terms in a particular question, based on the historical documents created by the users. First, the term probabilities of the users are computed and smoothed according to the profiles of users. Then the probability of generating a question is obtained through multiplying probabilities of each term in the question.
2. **profile-based method:** The profile-based method combines the language models and the graph-based methods. First, the messages posted by a particular participant are considered as the profile of the user. Second, the probability of a user as a candidate of expert is calculated by multiplying the authority value, computed using the HITS algorithm from the post-reply network of threads, with the probability computed from the language model of the user.
3. **reply count:** Ranking according to reply count is to rank the users according to the numbers of threads users give replies to. This is a straight forward approach for predicting experts.

6.4 Generality Study

We first test the general effectiveness of our algorithm on the three PFs. The results are shown below in Table 6.

Table 6: Generality Study

Forum	*MAP*	*P@1*	*P@3*	*P@5*	*P@10*
Java Forum	0.73	0.58	0.6	0.65	0.73
Java Devshed	0.74	0.6	0.62	0.65	0.72
GEF	0.67	0.53	0.56	0.61	0.63

The results show that, on average, our algorithm can achieve about 70% on MAP and 65% on P@N in all the three PFs. The best precision `G-Finder` can achieve is about 74%, and the worst is about 53% on P@1. For the case of Top-1 users, the returned user set is small, which makes the percentage of experts small. However, `G-Finder` can still predict the right experts for about 60% of the times on average in such situations. The prediction results on two Java forums are a little higher than that on the *GEF Forum*. We think that, compared with the general Java programming problems, the problems in the *GEF Forum* is more difficult to answer and the percentage of users that give answer is relatively smaller, resulting lower MAPs and P@Ns on the *GEF Forum*.

6.5 Performance

In this experiment, we show the performance of our model compared with three other methods mentioned above on the three forums. The results are shown in Table 5.

The results show that, based on semantic information, `G-Finder` can improve the performance of expert searching by 21% on average on MAP, and 23% on P@10, compared to the profile-based model. The best case for MAP is on the *Java Forum*, and the improvement is about 25%. The worst case is on the *GEF Forum* with the improvement of 17%. From the result, we conclude that the `G-Finder` improves the precision of expert searching significantly.

6.6 Effectiveness of Concept Network

The main difference in our approach, as compared to the related work, is that we use the concept network in `G-Finder` to locate experts. So are the relationships among concepts really effective in improving the quality of prediction? In this experiment, we compare the prediction quality of `G-Finder` with (`G-Finder`) and without (Single Concept) the concept network.

The result, presented in Table 7, shows that using concept network improves precision by about 50% on MAP compared to Single Concept Method. For the top ten users returned, the use of concept networks returns two more recommended experts on average. Returning more experts can help to lower the work load of experts.

Table 7: Effectiveness of Concept Network

Method	*MAP*	*P@5*	*P@10*
G-Finder	0.73	0.65	0.73
Single Concept	0.50	0.45	0.49

6.7 Scalability

As the number of threads in forums is usually large, can G-Finder be applied on large data sets when the number of threads grows? Figure 8 shows the computation time taken by building both the concept networks and the user networks by G-Finder on data sets of different number of threads. The results show that the runtime of G-Finder is almost linear to the size of data set. It should be noticed that as the user networks grow with the size of the data set, the time cost used to compute the PageRank values of users will be a little longer. The linear time growth shows that our algorithm is scalable with the number of threads.

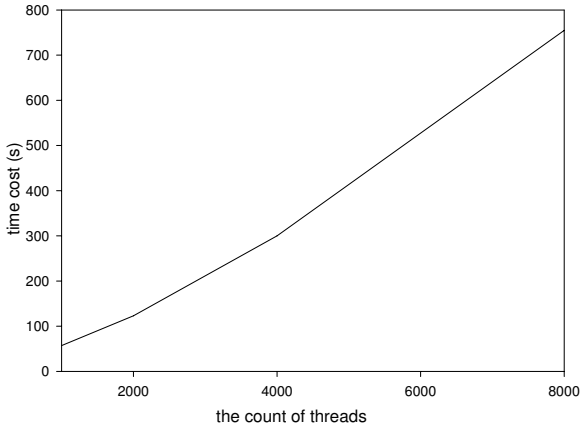

Figure 8: Time Cost/Count of Threads

We do not evaluate the query time. The probabilities, $P_{sep}(u|concept)$, and the likelihood probabilities, P, are computed and stored in the database after building the concept networks and the user networks. The cost of querying is mainly the time of mapping the query to the concepts, a tiny period of time that can be ignored.

6.8 Limitations of G-Finder

In this section, we discuss the limitation of G-Finder:

- *No Concepts in Threads:* Our heuristic-based concept mapping method in some cases fails to extract concepts. When these threads are in the test data set, our model can not return a reasonable result. However, in PFs, the percentage of threads that cannot be handled by our heuristics is usually low. Second, when these threads are in the set for building models, we can just remove these threads because the remaining threads in the set are of a healthy number to build our model.
- *Precision of Mapping from Thread to Concept:* In the algorithm, we map the threads to concepts using a heuristic-based approach. So the threads may be mapped to concepts they do not semantically belong to. We do not calculate the mapping precision in the experiment, but just calculate the final result. In fact, we can not quantify this error, as in the current forums no labeled mapping between threads and concepts exists. In some forums, a question is marked with tags, such as "java" or "perl". However, these tags are too high-level and not sensible to construct concept networks. So we just compute the final precision to evaluate the performance of our algorithm.

The probabilistic nature of our algorithm requires a significant amount of data to build both the user and the concept networks. Active forums usually have sufficient data for our algorithm to be applied.

6.9 Analysis of Prediction Errors

We carefully examined the evaluation results and found that G-Finder fails to find experts on the following three types:

1. **General Experts:** From the perspective of the concept network, we find that a certain type of users can be observed as "general experts", as the concepts of the questions they answer have no explicit or implicit semantic relationships with each other.
2. **Random Experts:** Another type of false positives is what we refer to as the random experts. They actively participate in the threads with comments and suggestions but rarely with answers. Our algorithm makes correct guesses that they are likely to give the answers but unable to distinguish between real answers and commentary texts.
3. **New Experts:** The third type of false positives belongs to users who never reply to anything before. Their first posts on the forums give the right answers to the questions. We refer to these users as new experts.

These three types of users do not have high rankings in our results, but provide the right answers in the verified data set. This is due to the special participation behavior of these users on the forums. As users are free to post anything to any threads, there are probably other latent patterns and relationships that are not captured by our model. We plan to address these issues in our future work.

7. Conclusions and Future Work

We have presented the design and the evaluation of G-Finder, both an algorithm and a tool that locates the most appropriate user of programming forums for answering a particular programming question. G-Finder makes use of the source code information of the target software system of a particular forum to discover additional latent relationships among forum threads and, consequently, significantly improves the quality of the expert search. We evaluated G-Finder using the real world data from active programming forums and compared our approach to the state of the art forum analysis techniques. As the future work, we plan to employ more sophisticated techniques to more accurately extract the semantic characteristics of the threads. We also