Routing Questions to the Right Users in Online Communities

Yanhong Zhou[1], Gao Cong[2], Bin Cui[1], Christian S. Jensen[2], Junjie Yao[1]

[1]*School of EECS & Key Laboratory of High Confidence Software Technologies, Peking University, China*
{yhzhou,bin.cui,junjie.yao}@pku.edu.cn

[2]*Department of Computer Science, Aalborg University, Denmark*
{gaocong,csj}@cs.aau.dk

***Abstract*— Online forums contain huge amounts of valuable user-generated content. In current forum systems, users have to passively wait for other users to visit the forum systems and read/answer their questions. The user experience for question answering suffers from this arrangement. In this paper, we address the problem of "pushing" the right questions to the right persons, the objective being to obtain quick, high-quality answers, thus improving user satisfaction. We propose a framework for the efficient and effective routing of a given question to the top-k potential experts (users) in a forum, by utilizing both the content and structures of the forum system. First, we compute the expertise of users according to the content of the forum system—this is to estimate the probability of a user being an expert for a given question based on the previous question answering of the user. Specifically, we design three models for this task, including a profile-based model, a thread-based model, and a cluster-based model. Second, we re-rank the user expertise measured in probability by utilizing the structural relations among users in a forum system. The results of the two steps can be integrated naturally in a probabilistic model that computes a final ranking score for each user. Experimental results show that the proposals are very promising.**

I. Introduction

A forum system is an online web application that enables a forum for discussions among users and for the posting of user-generated content. In particular, many forums serve as question answering portals where users in online communities ask and answer questions. A large number of such forums exist, and many forums can easily have thousands of users and thousands of questions posted per day. In addition, Community-based Question-Answering (CQA) portals, e.g., Yahoo! Answers (http://answers.yahoo.com/), Live QnA (http://qna.live.com/), and Baidu Zhidao (http://zhidao.baidu.com), can be regarded as variations of online forums. Since their inception, they have rapidly gained popularity. For example, Yahoo! Answers has attracted millions of users and hosts a huge number of questions. The question answering content in online communities provides an alternative for users to obtain information in the form of answers authored by other users, rather than as lists of results or documents from search engines.

With existing forum systems, users must passively wait for other users to visit the forums, read their questions, and provide answers. It may take hours or days from asking a question in a forum before a user can expect to receive answers. A user who knows well the answer to a particular question may not answer the question because the user may not visit the forum frequently or the user may be faced with many open questions. On the other hand, a user who answers a question may just happen to see the question, but is not an expert on the question's subject. To improve on this arrangement, we propose to push questions to the right persons in a forum system to obtain quick, high-quality answers. The reduced waiting times and improvements in the quality of answers are expected to improve user satisfaction.

Conventional forum system can be extended easily to support mobile communities, where the users ask and answer questions via mobile phones. The push mechanism is essential in a mobile CQA system. Consider a scenario where a user who is driving with his family from Hamburg to Copenhagen asks a question on a mobile CQA forum by sending a text message "Can you recommend a place where my kids, ages 4 and 7, can have good food and can play near the Copenhagen railway station?" Here the user definitely hopes to receive answers as soon as possible, and a quick reply is essential for such a service. If the CQA system does not have any answer that matches the user's question well, it can send the question to the right experts, who can provide the user with answers.

To enhance forum systems with the proposed push mechanism, we address the problem of effectively and efficiently finding the right users for a given new question in a forum. We believe that the resulting solution can significantly improve user satisfaction.

Previous work on ranking experts in forum systems [7], [20] focuses on computing a global ranking of users in a system by adapting graph-based ranking algorithms, such as the PageRank and HITS algorithms. They make use of the question-reply structural information of forums, but not the content of the posts. Furthermore, it will not be appropriate to route questions on different topics to the top users in a global ranking of the expertise of users in a forum system. Another line of related work, on expert search in enterprise documents [3], [4], assumes a setting where a set of documents, a list of candidate experts, and a set of topics are given and then finds experts for each of the given topics. This differs from our setting where we are to find the right experts for any new question without being given a predefined topic. In the above solutions, experts

The work was partially supported by NSF China under Grant No.60603045.
Contact author: bin.cui@pku.edu.cn

are found using plain documents, while forum data have a question-reply structure—such structural information between users can be useful in discriminating their relative areas of expertise. We also note that work on expert search tends to focus on effectiveness rather than efficiency.

To address the aforementioned problems, we propose a framework to effectively and efficiently find the top-k potential experts (users) for a given question in a forum. A forum contains a number of *threads*, each of which usually has a *question* post and a number of *reply* posts. To facilitate this task, we utilize both the content and structures of a forum. Specifically, we split the task into two. First, we compute the expertise of users according to the content of the forum, i.e., we estimate the probability of a candidate user being an expert for a given question based on the previous question answering activities of the user. Second, we re-rank the user expertise measured in probability by utilizing the structural relations among users in the forum. Specifically, we adopt graph-based ranking algorithms [20] to promote users with high authority. The results of the two sub-tasks are then integrated naturally into a probabilistic model capable of computing a final ranking score for each user.

We propose three different approaches to compute the user expertise (the first sub-task), namely a profile-based, a thread-based, and a cluster-based model.

1) **Profile-based model:** To compute the probability of a user being an expert for a given question, we create a profile for each user that represents the user's knowledge based on the answers authored by the user and also the corresponding questions he/she answered.
2) **Thread-based model:** In this approach, each thread serves as a latent topic. We thus compute the probability of a user being an expert for a new question based on each thread and the association between the thread and relevant users. Intuitively, we obtain a user profile for each thread, and each thread-based profile contributes to the ranking score of a user on the basis of the association of the thread with users.
3) **Cluster-based model:** We group threads with similar content into clusters and build a cluster-based thread for each user. Each cluster represents a coherent topic and is associated with users that indicates the relevance between the cluster and users. Given a new question, we compute the ranking score for each user by aggregating all clusters.

It is computationally expensive to calculate the ranking values with the three models, especially since multiple users may pose questions to a forum system simultaneously. To achieve better performance, we build inverted indexes for user profiles in the profile-based model, thread-user profiles in the thread-based model, and cluster-user profiles for the cluster-based model. For query processing, we adapt the well-known Threshold Algorithm [5] to efficiently access the inverted lists.

Re-ranking is accomplished using the structural relations between users in a forum. Such relations are formed naturally if a user replies to a question from another user. The question-reply network suggests a relative expertise levels between users. In contrast to the PageRank algorithm [12] that gives the same weight to all links, we assign a weight to each edge based on the the frequency of one user replying to another. For both the profile-based and thread-based approaches, we use all the threads in a forum to build a user network, and we adapt the PageRank algorithm to compute an authority value for each user. In contrast, in the cluster-based approach, the clusters serve as the unit for building a weighted question-reply network. The re-ranking scores computed from the threads in a cluster reflect the authority of the users in the cluster.

In summary, the paper makes the following contributions.

- We propose a push mechanism for online forums (including Community-based QA systems). To support this mechanism, we propose novel approaches to find experts for new questions in forums. To the best of our knowledge, this is the first work to address this problem.
- We extend the threshold algorithm for the query processing of our models to achieve higher efficiency.
- We conduct a detailed experimental study using a real-life forum data set. We evaluate the performance of our proposals and also compare with two baseline approaches. The experimental results show that our approaches yield satisfactory performance and significantly outperform the baseline approaches.

The rest of this paper is organized as follows. Section II reviews related work. Section III details the proposed framework including models and algorithms. Section IV reports on the performance study, and finally we conclude this paper in Section V.

II. Related Work

Online communities, e.g., forums, are often organized into sub-forums, each of which contains a number of threads; each thread usually contains a question post and a number of reply posts. Every person in a forum can pose a question and answer a question, which means that non-expert users might give incorrect answers. We note that no forum system or CQA service automatically supports the push mechanism proposed in this paper. The web service allexperts.com allows users to manually choose among registered experts and to ask questions of the selected experts by email.

Expert finding in social communities, such as forums, has attracted some research attention recently [7], [20]. Zhang et al. [20] use the network-based ranking algorithms HITS and PageRank to rank users in a "Java Forum" based on their authority scores. In other work [7], a similar approach is applied to a small data set obtained from the community based QA service Yahoo! Answers. The ranked list of users generated in these proposals may be used by the push mechanism proposed in this paper, although these proposals are not designed for this purpose and do not consider the content of online communities. As will be shown in our experimental results, the performance of using the ranked list for the push mechanism is poor. Although the ranked list alone is not

sufficient for the push mechanism, the network-based approach can be integrated into this paper's framework.

Our work is also related to expert search [10] in scientific and enterprise data. A topic model has been proposed for capturing reviewer's expertise based on the papers they have written [11]. The interest in expert finding is prompted by the launch of the Enterprise track of TREC [17], [6]. Language models are the dominating techniques used in expert search for enterprise data (e.g., [3], [13]). Language models have sound foundations in statistical theory and have performed quite well empirically in many information retrieval tasks [15], [19].

The language model-based approaches can be divided into profile-based methods and document-based methods. The profile-based methods (e.g., [3]) model the knowledge of an expert from associated documents with profiles and rank the candidate experts for a given topic based on the relevance scores between their profiles and the given topic. The document-based methods (e.g., [3], [14]) find related documents for a given topic and rank the candidates based on mentions of the candidates in the related documents. A hierarchical language model proposed by Petkova and Croft [13] uses a more fine-grained approach with a linear combination of the language models built on subcollections of documents. Our work differs from the existing work on expert search in three key respects. First, the existing techniques are mainly designed for plain documents, not for forum threads containing the question-reply structure and involving multiple users. Second, language models are established on top of term frequencies, and it has been shown that expert search relying only on word and document frequencies is limited [8] because it does not discriminate the relative expertise levels of users well; in forums, the network structure on the users can be utilized. Third, existing work on expert search usually ignores the efficiency aspect.

III. Our Approach

In this section, we present our approach to address the problem of routing questions to the right users in forum systems. As discussed previously, the existing work on expert search/ranking [3], [4], [7], [20] cannot be deployed effectively in this scenario due to the characteristics of forums. Therefore, we propose a new framework to find the top-k users for a given question by utilizing the content and structure of forums, which is illustrated in Figure 1. Our approach consists of two components: The expertise model that ranks the users according to their expertise captured by language models and the re-ranking model that re-ranks the candidate users using the question-reply graph.

A. The System Framework

Figure 1 shows the framework of our approach. The expertise model in the framework serves to find promising experts (users). Given the training thread data, we build an expertise index that captures the relation between users and the contents that they posted in the forum. Different models are designed in this paper to present the expertise of users. Additionally, we design a re-ranking model that uses the structure of forums to compute the authority of users to improve the user ranking. Given a new question, we compute the user expertise on the question using the expertise models, get the authority score of the users, rank the users based on their expertise and authority score, and return a ranked list of users who have high probabilities of being experts for the question.

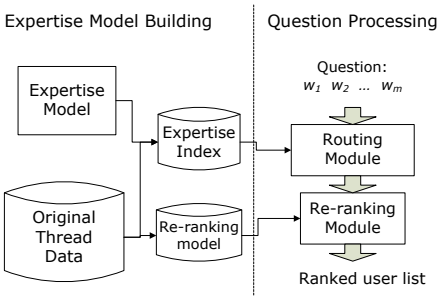

Fig. 1. Framework of routing question to users.

In this paper, the expertise of users and the authority of users are unified by means of a probability model. Given a new question q, the probability of a user u being an expert on the question is estimated as follows:

$$p(u|q) = \frac{p(q|u)p(u)}{p(q)}\ , \tag{1}$$

where $p(u)$ is the prior probability of a candidate user and $p(q)$ is the probability of a question generated by a random user, which is the same for all candidate users. In our approaches, $p(u)$ is estimated by the authority of user u, while $p(q|u)$ is used to capture the expertise of user u on question q. In this study our task is therefore to compute the probability $p(q|u)$ and $p(u)$.

This probability model was first introduced in [3]. However, $p(u)$ was assumed to be uniformly distributed and candidate users were ranked only according to $p(q|u)$.

B. Ranking Users with Language Model

In this subsection, we present three approaches to compute the expertise of the users according to their previous activities in forum systems. Given a new question q, users will be ranked based on the expertise model, i.e., $p(q|u)$.

According to the data characteristics of forums, we design three different models based on a language model to represent the users' expertise, namely a profile-based model, a thread-based model, and a cluster-based model.

B.1 Profile-based Model

In this model, a candidate user u is represented by a multinomial probability distribution over the vocabulary of words (i.e. $p(w|\theta_u)$). A new question q is represented by a set of words, and each question word is assumed to be generated independently. Therefore, the probability of question q being generated by the profile model can be obtained by taking the product across all the words in the question:

$$p(q|u) = \prod_{w \in q} p(w|\theta_u)^{n(w,q)}, \quad (2)$$

where $p(w|\theta_u)$ is the probability of a word given the profile model θ_u, and $n(w, q)$ is the number of times word w occurs in q. The core component of the above equation is to compute $p(w|\theta_u)$. In our approach, we first compute a raw profile $p(w|u)$ for user u by marginalizing over all threads:

$$p(w|u) = \sum_{\mathbf{td}} p(w|\mathbf{td}_u) con(\mathbf{td}, u), \quad (3)$$

where $p(w|\mathbf{td}_u)$ represents the probability of a word w generated by the thread $\mathbf{td}$ with the user u and $con(\mathbf{td}, u)$ is the contribution of user u on the thread $\mathbf{td}$. It is nontrivial to compute these two factors due to the structure of threads, and we will discuss them later in this section.

To obtain an estimate of the profile-based model $p(w|\theta_u)$, we need to do smoothing on the above $p(w|u)$. The smoothing is needed because many words in the vocabulary will not appear in a given user profile. Taking smoothing as a standard practice [19], we can avoid assigning zero probability $p(w|\theta_u)$ to unseen words and also make the estimated model more accurate. Specifically, we apply the Jelinek-Mercer method to smooth $p(w|u)$ and thus obtain the profile model $p(w|\theta_u)$ as follows:

$$p(w|\theta_u) = (1 - \lambda) p(w|u) + \lambda\ p(w) \quad (4)$$

where $p(w)$ denotes the background language model built on the entire collection C (can be all threads in a forum) and $\lambda \in [0, 1]$ is a coefficient to control the influence of the background model. The background model $p(w)$ is computed through a maximum likelihood estimation:

$$p(w) = \frac{n(w, C)}{|C|}, \quad (5)$$

where $n(w, C)$ denotes the frequency of word w occurring in the collection C and $|C|$ is the number of all words in collection C.

B.1.1 Language Models Built on Thread Data

We give two language modeling methods to model the content of one thread $\mathbf{td}$ with user u, $p(w|\mathbf{td}_u)$.

Single-doc thread model. We simply concatenate the question post $\mathbf{q}$ and the reply $\mathbf{r}_u$ from user u into a single document. We then compute the maximum likelihood estimation of each word w in the thread $\mathbf{td}$.

$$p(w|\mathbf{td}_u) = \frac{n(w, \mathbf{q}) + n(w, \mathbf{r}_u)}{|\mathbf{q} \cup \mathbf{r}_u|} \quad (6)$$

Question-reply thread model. The *single-doc* thread model does not distinguish the question and the reply. In contrast, the *question-reply* thread model gives different weights to the question post $\mathbf{q}$ and the reply $\mathbf{r}_u$. That is, we build a hierarchical language model [13] for a thread using its question-reply structure. If u has more than one reply in the thread $\mathbf{td}$, we combine all the replies into one reply.

$$p(w|\mathbf{td}_u) = (1 - \beta) p(w|\mathbf{q}) + \beta\ p(w|\mathbf{r}_u) \quad (7)$$

where $p(w|\mathbf{q})$ and $p(w|\mathbf{r}_u)$ denote the maximum likelihood estimation of word w in the question post $\mathbf{q}$ and the reply $\mathbf{r}_u$, respectively, for the current thread $\mathbf{td}$ with user u. The coefficient $\beta \in [0, 1]$ is used to enable trade-off between the question post $\mathbf{q}$ and the reply $\mathbf{r}_u$.

B.1.2 Modeling A User's Contribution to A Thread

We need to estimate the association between a thread $\mathbf{td}$ and a candidate user u. This association can be taken as the contribution of the user u to thread $\mathbf{td}$ in solving the question of this thread. We denote the contribution as $con(\mathbf{td}, u)$.

For the thread data, some replies in a thread may be better than other replies in the same thread. Ideally, we should take the quality into account when we evaluate the contribution of a reply to a question post. However, it is difficult to determine the quality of a reply. In this paper, we give a language model based approach to estimating the contribution of a reply. This is based on two observations: 1) it is shown that the question and answer often share some common words, and 2) answers from different users may often share similar words.

In this contribution model, we measure the contribution of a reply in terms of the likelihood of the question post to the reply, i.e., $p(\mathbf{q}|\theta_{\mathbf{r}_u})$[1].

To obtain probabilities, we normalize by the sum of the contributions of a specific user to all relevant threads. Given a user u, a thread $\mathbf{td}$ with question post $\mathbf{q}$ and a reply $\mathbf{r}_u$ authored by user u, the contribution of u on thread $\mathbf{td}$ is defined by:

$$con(\mathbf{td}, u) = \frac{p(\mathbf{q}|\theta_{\mathbf{r}_u})}{\sum_{\mathbf{td}'} p(\mathbf{q}'|\theta_{\mathbf{r}'_u})} = \frac{\prod_{w \in \mathbf{q}} p(w|\theta_{\mathbf{r}_u})}{\sum_{\mathbf{td}'} \prod_{w \in \mathbf{q}'} p(w|\theta_{\mathbf{r}'_u})}, \quad (8)$$

where $\theta_{\mathbf{r}_u}$ is a smoothed language model built on the reply by candidate user u in thread $\mathbf{td}$. Assume we take $p(w|\mathbf{r}_u)$ as the maximum likelihood estimation for word w in the reply $\mathbf{r}_u$, i.e., $\frac{n(w, \mathbf{r}_u)}{|\mathbf{r}_u|}$, and $p(w)$ as the background language model estimated by Equation 5, then $\theta_{\mathbf{r}_u}$ can be obtained as follows:

$$p(w|\theta_{\mathbf{r}_u}) = (1 - \lambda)\ p(w|\mathbf{r}_u) + \lambda\ p(w) \quad (9)$$

Comments: The proposed profile-based model differs from that of Balog et al. [3] for the method of estimating the language model $p(w|\theta_u)$ in two aspects. First, Balog et al.'s language model is estimated from a set of documents associated with a user u. In our problem, this is different since each user is associated with a set of replies, and each reply is associated with a question post. We build a hierarchical language model for each thread by utilizing the question-reply structure of a thread. And each thread may be associated with different

[1]In experiments, we utilize the logarithm of likelihoods rather than the likelihoods to avoid zero values.

users who have different thread language models, denoted as $p(w|\mathbf{td}_u)$. Second, to compute the contribution of a user u to a thread $\mathbf{td}$, we consider the content similarity between the question post and the user's reply, while Balog et al. connect a user with a document if the user occurs in the document.

B.1.3 Algorithms

It would be computationally expensive to compute the probability of a user being an expert for a question from scratch using the framework based on the profile-based model. To speed up expert search for a question, we pre-compute the language model and build an index. More specifically, expert search is done in two stages: **index creation** and **question processing**. In the index creation stage, we create an inverted list of $(u, p(w|\theta_u))$ for each word w. Each inverted list is sorted by the weight value of $p(w|\theta_u)$, which represents the relation between the word and corresponding user. Thus, in the question processing stage, we can apply the Threshold Algorithm [5] to compute the top-k ranked users who have the highest probability of being experts for the question.

Index Creation: Under our profile-based model, all the users have a profile represented as $p(w|\theta_u)$, and the expert search process is based on the users' profiles. In this stage, we need to compute $p(w|\theta_u)$ for all users.

In order to apply the threshold algorithm to speed up the query processing, for each word w, we build an inverted list of $(u, p(w|\theta_u))$ and sort it by the weight of $p(w|\theta_u)$. Figure 2 shows an inverted list example for our profile model. Here the inverted lists for words are sorted by the the value of $p(w|\theta_u)$.

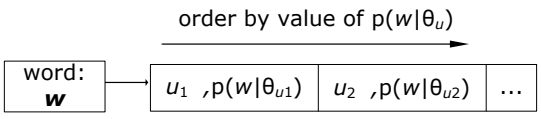

Fig. 2. Index structure for the profile-based model.

The inverted list creation process is shown in Algorithm 1. During the indexing process, we compute $p(w|\theta_u)$ for all users u and then sort the list of $(u, p(w|\theta_u))$. Finally, each word w has an inverted list, in which entries are sorted by the weight of $p(w|\theta_u)$.

Question Processing: Based on the sorted inverted lists built during index creation, we apply the threshold algorithm to compute the top-k ranked results for a new question. This algorithm achieves the least accesses to the inverted list for the computation of the top-k ranked result. Given a question query q containing l words, where each word w_i has the frequency $n(w_i, q)$ and corresponds to the sorted list $L_i = \{(u, p(w_i|\theta_u))\}$, the querying process using the threshold algorithm is illustrated as follows:

1) Let $Y = \{(u, score(u))\}$ keeps track of the current top k results in descending order of $score(u)$.
2) Conduct a sorted access to each of the l sorted lists L_i (i.e., access the top member of each of the lists under sorted access, then access the second member of each list, etc.). As an entry of $(u, p(w_i|\theta_u))$ is seen under sorted access in some list, conduct a random access to the other lists to find all other weights of $p(w_{-i}|u)$, where w_{-i} represents all other words in the question except for word w_i. Compute the score for u: $score(u) = \prod_{w \in q} p(w|\theta_u)^{n(w,q)}$. If Y is not full or $score(u)$ is larger than the minimal score in Y then store the pair $(u, Score(u))$ in Y.
3) For each list L_i, let $(u_i^*, p(w_i|\theta_{u_i^*}))$ be the last entry seen under the sorted access. Compute the threshold value t as $t = \prod_i p(w_i|\theta_{u_i^*})^{n(w_i,q)}$. If the scores of all the k users in Y are no less than t then stop.
4) Output the top-k results in set Y.

Algorithm 1 Index creation of profile-based model

```
1: //Generation stage
2: for each user u do
3:    For each word w, initialize p(w|u) to 0;
4:    Find all threads {td} replied to by user u and compute
      con(td, u);
5:    for each td in {td} do
6:       for each word w in td do
7:          Consider the question and the reply of u in td to
            compute p(w|td_u);
8:          p(w|u) += p(w|td_u) con(td, u);
9:       end for each word
10:   end for each thread
11:   Smooth all p(w|u) to p(w|θ_u);
12:   For each word w, store the triplet of (w, u, p(w|θ_u))
13: end for each user
14: // Sorting stage
15: for each word w do
16:   Find the list of (u, p(w|θ_u)) and sort it by p(w|θ_u);
17:   Store the sorted list of (u, p(w|θ_u)) for w;
18: end for each word
```

In the above querying process, we do not need to compute the scores for all users, which can speed up the computation of the top-k expert users.

B.2 Thread-based Model

In this model, we assume that each thread is a latent topic. Each thread $\mathbf{td}$ is assigned to a user u with the probability $con(\mathbf{td}, u)$, which denotes the contribution of the user u to the thread $\mathbf{td}$. The model built on the threads, i.e., the $p(w|\theta_{\mathbf{td}})$ act as small-grain profiles for each user. Unlike the document model of Balog et al. [3], we build a hierarchical language model on each thread using its question-reply structure. Additionally, the contribution of a user to each thread is estimated by the content similarity between the question and the reply authored by the user.

We can also build two kinds of models for each thread, i.e., a *single-doc* thread model and a *question-reply* thread model as in Section III-B.1.1. Different from the profile-based model, we combine all the replies of a thread into one reply, but do not distinguish the replies from different users. The reason for doing this is that it will be too computationally expensive to have a separate language model for each combination of a user and a thread. Specifically, we estimate $p(w|\mathbf{td})$ by applying the Equation 6 or Equation 7, and we obtain the smoothed

model $p(w|\theta_{\mathbf{td}})$ after smoothing with the background model $p(w)$.

$$p(w|\theta_{\mathbf{td}}) = (1-\lambda)p(w|\mathbf{td}) + \lambda\ p(w) \tag{10}$$

Under this model, the probability of generating a new question q can be viewed as the following generative process. For each user, we choose a latent topic, i.e., thread, at the probability estimated by the contribution of the user to the thread $con(\mathbf{td}, u)$; from the latent topic, we generate the given new question with probability $p(q|\theta_{\mathbf{td}})$. Another way to view the generative process is to build a profile-based model for each thread. By summing over all threads, we obtain the probability of question q being generated by the thread-based model as formally computed below.

$$p(q|u) = \sum_{\mathbf{td}} p(q|\theta_{\mathbf{td}})con(\mathbf{td}, u), \tag{11}$$

where $con(\mathbf{td}, u)$ is computed using Equation 8 and $\theta_{\mathbf{td}}$ is the smoothed language model built on thread $\mathbf{td}$.

To compute the probability $p(q|\theta_{\mathbf{td}})$ of a question q given a thread $\mathbf{td}$, we take the product of generating a word by $\mathbf{td}$ across all the words in the question q.

$$p(q|\theta_{\mathbf{td}}) = \prod_{w \in q} p(w|\theta_{\mathbf{td}\_u})^{n(w,q)} \tag{12}$$

B.2.1 Algorithms

The algorithm for the thread-based model is implemented in two stages.

Index Creation: We need two kinds of inverted lists for this model: *thread list* and *thread user contribution list*. *Thread list* is used to store the user's profile specific to a certain thread, i.e., $p(w|\theta_{\mathbf{td}})$, and *thread user contribution list* is used to store the contribution of user to the thread, i.e., $con(\mathbf{td}, u)$.

Figure 3 shows the index structure example used for this model. Actually, QA systems providing question or answer search (or a search engine) usually has an index such as the *thread list*, and we could reuse the existing index structure, avoiding to build *thread list*. In other words, the thread-based model only needs to build an additional *thread user contribution list* for routing questions to the right users. This is also an advantage of the thread-based model.

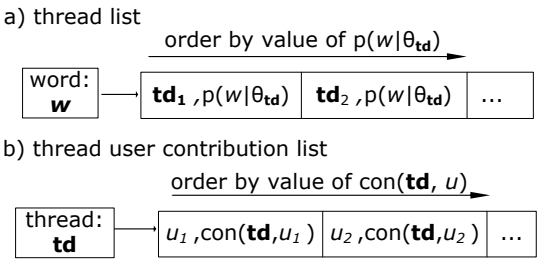

Fig. 3. Index structure for the thread-based model

In order to apply the threshold algorithm for query processing, we build for each word an inverted list of $(w, p(w|\theta_{\mathbf{td}}))$ and sort it by the value of $(p(w|\theta_{\mathbf{td}}))$. Furthermore, we also build a sorted list of $(u, con(\mathbf{td}, u))$ for each thread $\mathbf{td}$. The index building process is given in Algorithm 2.

Algorithm 2 Index creation for the thread-based model

```
// Generation stage
for each thread td do
    Combine the replies in td into one reply r;
    for each word w do
        Consider the question q and reply r in td to compute p(w|td);
        Smooth p(w|td) to p(w|θ_td);
        Store the triplet (w, td, p(w|θ_td));
    end for each word
end for each thread
for each user u do
    Find all threads {td} replied to by u and compute con(td, u);
    For each td in {td}, store the triplet of (td, u, con(td, u));
end for each user
// Sorting stage
for each word w do
    Find the list (td, p(w|θ_td) and sort it by p(w|θ_td);
    Store the sorted list of (td, p(w|θ_td) for w;
end for each word
for each thread td do
    Find the list (u, con(td, u)) and sort it by con(td, u);
    Store the sorted list of (u, con(td, u)) for td;
end for each thread
```

Question Processing: Given a question query, we can make use of the index structure in Figure 3 to compute expertise score for all users. However, this remains computationally expensive since we need access a large number of inverted lists. We next extend the threshold algorithm to approximately compute the top-k ranked results under this model.

We separate the query processing into two stages. For a given question query q, the relevant threads $\mathbf{td}$ with the score of $p(q|\theta_{\mathbf{td}})$ are found in the first stage. Existing QA systems provide this service. In our work, we apply the threshold algorithm to the *thread list* shown in Figure 3 to find the rel threads that are the most similar to a given question query q, where rel is a parameter. This process is similar to the one where we use the threshold algorithm to compute the top-k results in the profile-based model. The difference is, that we now need to compute the most relevant threads with a score of $p(q|\theta_{\mathbf{td}})$. In the second stage, we apply these most relevant threads, e.g., rel threads, and access the *thread user contribution list* to compute the top-k users according to Equation 11. We apply the threshold algorithm in this stage to compute the top-k ranked users for the question q.

Specifically, let a new question query q containing l words $w_1, w_2, ..., w_l$ together with the frequency $n(w_i, q)$ of each word w_i be given together with a sorted *thread list* $L'_i = \{(\mathbf{td}, p(w_i|\theta_{\mathbf{td}}))\}$ of each word w_i. Then, the process for approximately computing the top-k ranked users under this model proceeds as follows:

1) First stage: for question q, find the rel threads $\{\mathbf{td}\}$ with the highest relevant score $p(q|\theta_{\mathbf{td}})$, where rel is a parameter to be set empirically.

a) Let $Y' = \{(\mathbf{td}, score(\mathbf{td}))\}$ keeps track of the top rel threads in descending order of $score(\mathbf{td})$.
b) Conduct sorted access to the l sorted thread lists L'_i in parallel. As an entry of $(\mathbf{td}, p(w_i|\theta_{\mathbf{td}}))$ is seen under sorted access in some list, do random access to all other thread lists and find all the weights of $p(w_{-i}|\theta_{\mathbf{td}})$ for the current thread $\mathbf{td}$. Here w_{-i} represents all other words in question q except word w_i. Compute the score of thread $\mathbf{td}$ as follows: $score(\mathbf{td}) = \prod_{w \in q} p(w|\theta_{\mathbf{td}})^{n(w,q)}$. If Y' is not full or the score of $\mathbf{td}$ is larger than the minimal score in Y' then store the record of $(\mathbf{td}, score(\mathbf{td}))$ in Y'.
c) For each thread list L'_i, let $(\mathbf{td}^*_i, p(w_i|\theta_{\mathbf{td}^*_i}))$ be the last entry seen under sorted access in the list of L'_i. Compute the threshold value t' as $t' = \prod_i p(w_i|\theta_{\mathbf{td}^*_i})^{n(w_i,q)}$. As soon as the scores of all rel threads in Y' are no less than t', stop.

2) Second stage: based on those rel threads in $Y' = \{\mathbf{td}_1, \mathbf{td}_2, ..., \mathbf{td}_{rel}\}$, compute the top-$k$ ranked users for a given question. Each thread $\mathbf{td}_i$ in Y' corresponds to a sorted *thread user contribution list* $L_i = \{(u, con(\mathbf{td}_i, u))\}$.
a) Let $Y = \{(u, score(u))\}$ keeps track of the top k users in descending order of $score(u)$.
b) Conduct sorted access to the rel sorted contribution lists L_i in parallel. As an entry of $(u, con(\mathbf{td}_i, u))$ is seen under sorted access in some list, do random access to all other lists and find the weights of $con(\mathbf{td}_{-i}, u)$ for the current user u. Here $\mathbf{td}_{-i}$ denotes all other thread in Y' except thread $\mathbf{td}_i$. Compute the score of user u as follows:
$$score(u) = \sum_{\mathbf{td} \in Y'} score(\mathbf{td}) con(\mathbf{td}, u).$$
If Y is not full or the score of u is larger than the minimal score in Y then store the record of $(u, score(u))$ in Y.
c) For each contribution list L_i for thread $\mathbf{td}_i$, let $(u^*_i, con(\mathbf{td}_i, u^*_i))$ be the last entry seen under sorted access in the contribution list of L_i. Compute the threshold value t as $t = \sum_i score(\mathbf{td}_i) con(\mathbf{td}_i, u^*_i)$. As soon as the scores of all k users in Y are no less than t, stop.

Note that we apply the threshold algorithm in both stages, one for finding relevant threads and one for finding the relevant users. Thus, we do not need to scan all threads and users. Additionally, for efficiency, we only consider the top rel number of threads to compute the scores of users.

B.3 Cluster-based Model

In this model, thread clustering is used to group posts according to similar content (topic). Each cluster is assumed to represent a topic, and to contain only threads related to that topic. Language models are estimated for the clusters, and we build a smoothed language model on each cluster, i.e., $p(w|\theta_{Cluster})$.

Then, for a given new question q, the probability of q for a user u can be viewed as the following generative process. For each user, we choose the cluster with the probability estimated by the contribution of the user to the cluster; according to the cluster information, we generate the given new question q with probability $p(q|\theta_{Cluster})$. Another way to view the generative process is to build a profile-based model for each cluster. By summing over all clusters, we will obtain the probability given below of question q being generated by the cluster-based model.

$$p(q|u) = \sum_{Cluster} \prod_{w \in q} p(w|\theta_{Cluster})^{n(w,q)} con(Cluster, u), \tag{13}$$

where $p(w|\theta_{Cluster})$ represents the smoothed language model built on a cluster. To compute this model, we first combine the questions of threads in the cluster into a new question Q and the replies in the cluster into one reply R to question Q. That is, we view each cluster as a big "thread" Td with a question Q and a reply R. As in the case of the methods with which we build the thread language model in Section III-B.1.1, we estimate the cluster model $p(w|Cluster)$ as $p(w|Td)$ using either the *single-doc* model in Equation 6 or the *question-reply* model in Equation 7. Then we smooth this model by the background language model $p(w)$ to avoid the zero probability for unseen words as follows:

$$p(w|\theta_{Cluster}) = (1 - \lambda) p(w|Cluster) + \lambda\ p(w) \tag{14}$$

With this model, the probability that a latent topic, i.e., a cluster, is assigned to a user is estimated as the contribution of the user to the cluster, $con(Cluster, u)$. This quantity is the computed as the sum over the contributions of the user to all relevant threads within the cluster:

$$con(Cluster, u) = \sum_{\mathbf{td}} con(\mathbf{td}, u), \tag{15}$$

where the contribution of user to a thread, $con(\mathbf{td}, u)$, is computed according to Equation 8 in Section III-B.1.2.

Generating Clusters: We need to cluster threads into groups so that, ideally, all threads in a group are on a similar topic. We observe that forums are often organized into sub-forums, and we can use the sub-forums for generating clusters. We can also employ clustering to thread data to generate the clusters. Our proposal is equally applicable. The number of clusters is usually fixed and not very large.

Index Creation: As for the cluster-based method, we have two inverted lists for this method: the *cluster list* and the *cluster user contribution list*. Figure 4 illustrates the index structure for this method. The *cluster user contribution list* captures the association between users and clusters, while the *cluster list* is mainly for recording the contents of clusters. The index creation process is detailed by Algorithm 3.

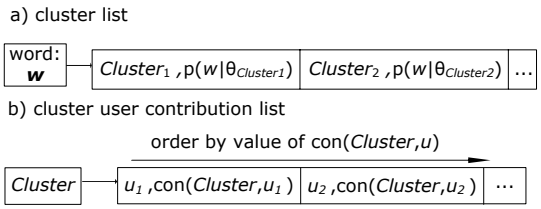

Fig. 4. Index structure for the cluster-based model

Algorithm 3 Index creation for the cluster-based model

```
1: // Generation stage
2: for each cluster do
3:    Combine all questions in the cluster into one question Q,
      combine all replies in the cluster into one reply R and consider
      current cluster as a pseudo thread Td;
4:    for each word w do
5:       Use Q and R to compute p(w|Td) and estimate
         p(w|Cluster) as p(w|Td);
6:       Smooth p(w|Cluster) to obtain p(w|θ_Cluster);
7:       Store (Cluster, p(w|θ_Cluster)) in the cluster list of w;
8:    end for each word
9: end for each cluster
10: for each user u do
11:    Find all the threads replied to by u and compute con(td, u);
12:    for each cluster do
13:       In the current cluster, find the threads {td} replied to by
          u;
14:       Initiate con(Cluster, u) as 0;
15:       for each td in {td} do
16:          con(Cluster, u) += con(td, u);
17:       end for each thread
18:       Store the triplet of (Cluster, u, con(Cluster, u));
19:    end for each cluster
20: end for each user
21: // Sorting stage
22: for each cluster do
23:    Find the list of (u, con(Cluster, u)) and sort it by
       con(Cluster, u);
24:    Store the sorted list of (u, con(Cluster, u));
25: end for each cluster
```

Question Processing: Based on the index structure in Figure 4, we compute the expertise score of all users for a given question query. Similar with the thread-based model, we separate the question processing into two stages. However, we only apply the threshold algorithm to the *cluster user contribution lists* in the second stage to compute the top-k ranked users for a given question query q.

For a given question query q, the first stage finds the relevant clusters with the score of $p(q|\theta_{Cluster})$ using the *cluster lists*. Then in the second stage, the threshold algorithm is applied to the *cluster user contribution lists* to compute the top-k users according to Equation 13.

Specifically, given a new question query q containing l words $w_1, w_2, ..., w_l$ and the frequency $n(w_i, q)$ of each word w_i, we compute the top-k ranked users as follows:

1) First stage: access the cluster lists and compute the score for each cluster as follows:
$$score(Cluster) = \prod_{w \in q} p(w|\theta_{Cluster})^{n(w,q)}.$$
2) Second stage: based on the scores of the clusters, access the *cluster user contribution lists* to compute the top-k ranked users; each cluster corresponds to a sorted *cluster user contribution list* $L_i = \{(u, con(Cluster_i, u))\}$.
   a) Let $Y = \{(u, score(u))\}$ records the top k users in descending order of $score(u)$.
   b) Conduct sorted access to the lists L_i in parallel. As an entry $(u, con(Cluster_i, u))$ is seen under the sorted access in some list, do random access to all other lists and find the weights of $con(Cluster_{-i}, u)$ for the current user u. Here $Cluster_{-i}$ denotes all other clusters except cluster $Cluster_i$. Compute the score of user u as follows: $score(u) = \sum_i score(Cluster_i) con(Cluster_i, u)$. If Y is not full or the score of u is larger than the minimal score in Y then store $(u, score(u))$ in Y.
   c) For each contribution list L_i of $Cluster$, let $(u_i^*, con(Cluster_i, u_i^*))$ be the last entry seen under sorted access in the contribution list of L_i. Compute the threshold value t as $t = \sum_i score(Cluster_i) con(Cluster_i, u_i^*)$. When the scores of all k users in Y are no less than t, stop.

C. Cost Analysis

We proceed to analyze the time complexity of index creation, then the index size, and finally the time complexity of query processing for our three approaches. We use the following notation: l represents the number of distinct words in the new question query q; d represents the number of threads in our data set; c represents the number of clusters in our data set; m represents the number of users in our data set; and n represents the number of distinct words in our data set.

Complexity of Index Creation: In all three approaches, index creation includes two stages: inverted lists generation and list sorting. For the list generation, the models all need approximately $O(ndm)$ time. In fact, most of time is used for computing the contributions of users to threads. Next, the time complexity of sorting lists is $O(nm \log m)$ for the profile-based model, which has n inverted lists of length m. Similarly, the list sorting time for the thread-based model and the cluster-based model is $O(nd \log d) + O(dm \log m)$ and $O(cm \log m)$, respectively.

Index Size: The index size is related to the number of lists and the sizes of the lists. The index space cost for the profile-based model is $O(nm)$. For the other models, we need two kinds of lists as explained before. The thread-based model needs $O(nd)$ space for the *thread list* and $O(dm)$ for the *thread user contribution list*. However, if an index on the thread data already exists in a QA system, the thread-based model can reuse this index and then only needs $O(dm)$ space for storing the *thread user contribution list*. The index size of cluster-based model is $O(nc) + O(cm)$.

Complexity of Query Processing: We consider the top-k retrieval time using the threshold algorithm. For the profile model, the worst case time complexity for computing the top-k users is $O(lm)$. The thread-based model first finds the relevant threads and then applies these to produce the final top-k users.

Assuming that we compute rel threads in the first stage, this model takes $O(ld)$ for this and takes $O(rel\ m)$, which is less than $O(dm)$, for top-k users. Similarly, for the cluster-based model, the top-k retrieval time is less than $O(lc) + O(cm)$.

D. Re-ranking Using the Question-Reply Structure

The thread data in a typical forum generally consists of question and reply posts, and each post is associated with a user. The question-reply structure can be used to assign authority to users.

1) Graph Building: In forum threads, the initial post usually contains a question, and reply posts are assumed to contain answers to the question in the initial post. Using the question-reply structure of posts, we can build a question-reply network (graph) for users by following an approach detailed in the literature [20]. Each user corresponds to a vertex in the graph, and a directed edge from u to v is generated if user v answers at least one question from user u. The weight of the edge is estimated by the frequency of user v replied a question from user u. The generated question-reply graph is not simply a social network reflecting connections among users. Importantly, one user replying to another user's question usually indicates that the former user is more knowledgable on the subject of the question, so if a user has many incoming edges, the user is perhaps an authority in the forum.

2) Expert Re-ranking: We adapt the PageRank algorithm [12] to our question-reply graph, and the rank values obtained are taken as the authority of users. Here we denote the authority of users by $p(u)$ and take this as the prior probability for user u to be an expert on a new question q. In Section III-B, we present three language model based methods of computing the probability of users being experts on a new question $p(q|u)$. In our re-ranking process, we take the product of $p(q|u)$ and $p(u)$ to rank the users for a specific new question q.

Specifically, we have two different re-ranking mechanisms for the different models. For the profile-based model and the thread-based model, we get the authority of users using all threads in our data set, i.e., $p(u)$ for each user, and then we re-rank the users according to $p(q|u)p(u)$. For the cluster-based model, we get the authority of users for each cluster. That is, for each user u, we have the authority score $p(u, Cluster)$ with respect to a specific cluster, and then we combine it with the cluster-based model as $\sum_{Cluster} p(q|Cluster)con(Cluster, u)p(u, Cluster)$. This score is used to re-rank the users for the cluster-based model.

IV. Empirical Study

In this section, we evaluate the question routing proposals in terms of both effectiveness and the efficiency.

We collected thread data from Tripadvisor forums (http://www.tripadvisor.com) and generated 6 data sets for evaluation. Statistics on these data sets is shown in Table I. The *BaseSet* is used for both effectiveness and efficiency evaluation, and the other 5 sets are used only for scalability evaluation. In the table, *#posts* denotes the number of question and reply posts in a data set, *#users* denotes the number of users having at least one reply post in a data set, *#words* denotes the number of distinct words in a data set, and *#clusters* denotes the number of sub-forums in a data set. All experiments were conducted on a computer with a 1.80GHz CPU, 1GB main memory and using Windows Server 2003.

TABLE I
THREAD DATA SETS

data set	#threads	#posts	#users	#words	#clusters
BaseSet	121,704	971,905	40,248	324,055	17
Set60K	60,000	337,656	37,088	228,639	17
Set120K	120,000	754,632	56,110	292,502	19
Set180K	180,000	1,092,288	88,522	447,139	19
Set240K	240,000	1,612,309	94,733	489,359	19
Set300K	300,000	1,949,965	125,015	629,229	19

In our experiments, we use the Lucene [1] to pre-process our thread data, including tokenization, stop words filtering, and stemming. After preprocessing, both the question post and replies of each thread are taken as bags of words. In addition, we employ Lucene to store the inverted lists that are the indexes for our approaches.

A. Effectiveness

We take the *BasesSet* dataset as our training set for evaluating the effectiveness of our 3 approaches.

1) Test Collection: It is difficult to evaluate the quality of the answers to a new question due to the scarcity of the evaluation data. Since no explicit question/user expertise relevance is available in Tripadvisor that can be used evaluation, we manually annotate the relevance between a new question and the users (candidate experts).

We selected 10 questions (not in *BasesSet*) as our new questions and randomly sampled 102 users, omitting users with fewer than 10 replies. For each sampled users, we collected the user's activity history in Tripadvisor as evidence of the user's expertise, including the questions and the user's replies to the questions. For the expertise relevance between a new question q and a user u, we define a 2-level relevance assessment scheme as follows:

- **(1):** User u has high expertise on the topic of question q. That is, user u has a number of high-quality replies on this topic.
- **(0):** User u has low expertise on the topic of question q.

Based on the above relevance assessment format, annotators were asked to judge all question/user pairs that we had selected. That is, we got 10×102 relevance assessments per annotator. We take these relevance assessments as the ground truth for the effectiveness evaluation.

2) Effectiveness Metrics: We apply the metrics used for the expert finding task in the *TREC Enterprise Track* [2] to evaluate the effectiveness of our approaches. The metrics include *Mean Average Precision* (MAP), *Mean Reciprocal Rank* (MRR) [18], [16], *Precision@N*, and *R-Precision* [9].

- **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 correct retrieved answer (expert), i.e. the precision after truncating the list after each of the correct answer (expert).
- **MRR**: MRR is the mean of the reciprocal ranks of the first correct answers over a set of query questions. While MAP considers all correct answers, this measure gives us an idea of how far down we must look in a ranked list in order to find a correct answer.
- **Precision@N**: Precision@N is the percentage of the top-*N* candidate answers retrieved that are correct.
- **R-precision**: It requires having a set of known relevant answers *Rel*, from which we calculate the precision of the top *Rel* answers returned.

3) Performance Tuning: As mentioned in Section III-B, we build different language models on thread data. Additionally, we have two parameters for all of our models: λ, which is used to smooth the language model, and β, which is used to specify the reply portion for the hierarchical *question-reply* thread model presented in Section III-B.1.1.

According to [19], using $\lambda \approx 0.7$ can produce optimal values for long queries. Consistent with this finding, our models can also obtain acceptable performance when $\lambda \approx 0.7$. The detailed results are omitted here.

Another factor that may affect the performance is how the language models are built on threads. Section III-B.1.1 presents two methods for building language models on thread data, i.e., the *single-doc* thread model and the *question-reply* thread model. Results for these are given in Table II. We can see that the *question-reply* model outperforms the *single-doc* model. The question-reply thread model, which builds a hierarchical language models on threads, can differentiate between questions and replies in contributions, which is appropriate in our scenario.

TABLE II
SINGLE-DOC V.S QUESTION-REPLY

Thread LM	MAP	MRR	R-Precision	P@5	P@10
Single-doc	0.567	0.761	0.391	0.54	0.54
Question-reply	**0.584**	**0.8**	**0.391**	**0.58**	**0.54**

Since the *question-reply* model obtains the better performance for our task, we proceed to tune parameter β, the coefficient determining the proportion of replies when building the model on a thread. The three models show similar behavior when varying β, and we only show the results for the thread-based model—see Table III. The experiments suggest that our models perform the best when $\beta = 0.5$.

TABLE III
EFFECTIVENESS OF DIFFERENT β FOR THREAD-BASED MODEL

Beta	MAP	MRR	R-Precision	P@5	P@10
0.3	0.566	0.766	0.382	0.56	0.53
0.5	**0.584**	**0.8**	0.391	**0.58**	**0.54**
0.7	0.576	0.747	**0.394**	**0.58**	0.53

The parameter rel might also influence the performance of the thread-based model. As discussed in the question processing part for the thread-based model, the computation of the top-k users proceeds in two stages, and we only use the top rel threads obtained from the first stage for calculating the top-k users. Using a lower rel would reduce the query processing time, but might also affect the correctness, since we miss some threads when computing the scores of users. So we are faced with a trade off between effectiveness and efficiency.

Table IV shows the effects of different rel on the performance of the thread-based model. We omit the results for MRR and Precision@10 because these do not vary when rel is varied. And "all" means we obtain all relevant threads in the first sub-stage and then apply all relevant threads to get the top-10 users for a given new question. We can see from the table that when $rel = 800$, the thread-based model obtains almost as good results as when all relevant threads are used, while needing significantly less time. Hence in our experiments, we use $rel = 800$ for the thread-based model.

TABLE IV
EFFECTIVENESS OF DIFFERENT rel FOR THE THREAD-BASED MODEL

rel	MAP	R-Precision	P@5	Top-10 search(second)
200	0.550	0.201	0.56	4.05
400	0.569	0.265	0.58	4.32
600	0.576	0.346	0.58	4.66
800	0.582	0.391	0.58	4.82
All	0.584	0.391	0.58	11.87

The following experiments consider the question-reply thread model and use the following values as default setting: $\lambda = 0.7$, $\beta = 0.5$, and $rel = 800$.

4) Effectiveness of Different Approaches: No previous work exists on routing new question to the right experts in forums. We thus compare our approaches with two baseline methods.

- *Reply Count*: This method uses the number of threads replied to by the user as the user's score.
- *Global Rank*: This method estimates the authority score of a user by the user's PageRank value in the question-reply graph [20].

Table V shows the results for our approaches and the two baseline models for the routing task. As shown in the table, our approaches all significantly outperform the baseline models. Thus simple statistical information and a global ranking score by structure information in a forum are insufficient for routing new questions to the right users. Neither consider the content.

TABLE V
EFFECTIVENESS OF THE DIFFERENT APPROACHES

Method	MAP	MRR	R-Precision	P@5	P@10
Replies Count	0.130	0.131	0.121	0.08	0.1
Global Rank	0.134	0.152	0.118	0.08	0.1
Profile	0.563	**0.87**	0.369	0.56	0.52
Thread	**0.582**	0.8	0.391	**0.58**	**0.54**
Cluster	0.532	0.736	**0.452**	0.46	0.49

Although the three proposed approaches exhibit different performance, the differences are not pronounced and there is

no a clear overall winner. The profile-based model yields the best performance for MRR. That is to say, the profile-based model ranks the first correct expert higher in the ranking list. This is understandable, since when we build the profile for all users, we consider the users' specific thread language models to build the users' expertise. This makes it is easier for the profile-based model to find an expert for a new question.

The thread-based model and the cluster-based model group the users according to latent topics, and users are connected either by the threads they have participated in or by clusters. When we route a new question to users, the ranking results by these two models are capable of yielding more experts. This may be because users are connected by latent topics, and if one user is returned, users who are experts on the same topic may also be returned. As shown in Table V, the thread-based model has the best performance for MAP, Precision@5, and Precision@10, and cluster-based model performs best for R-Precision.

For the comparison between the thread-based model and the cluster-based model, we can see that the thread-based model slightly outperforms the cluster-based model. The reason is that the thread-based model builds fine grained profiles for the users, i.e., the thread-based model builds profiles for the users on every thread, while the cluster model builds profiles for the users on every cluster (of which there are much fewer).

5) Effectiveness of Re-ranking: In our proposal, we apply the re-ranking module, which is based on the network analysis on the question-reply graph of our thread data. From Table VI, we can see that the re-ranking mechanism has only a marginal effect on some metrics; however, it does improve the performance in terms for MRR. The reason may be that the re-ranking algorithm is capable of promoting the active users with higher expertise to the top. Note that the MRR is important, as it measures how far down we must look in the ranked list in order to find a correct expert. In our application setting, the QA system should ideally send a question to only a few users who are experts on the question. In this sense, a high MRR is important.

TABLE VI
EFFECTIVENESS OF RE-RANKING FOR THE THREAD-BASED MODEL

Method	MAP	MRR	R-Precision	P@5	P@10
Profile	0.563	0.87	0.369	0.56	0.52
Profile+Rerank	0.569	**0.911**	0.344	**0.62**	0.47
Thread	**0.582**	0.8	0.391	0.58	**0.54**
Thread+Rerank	0.581	**0.911**	0.344	0.54	0.51
Cluster	0.532	0.736	**0.452**	0.46	0.49
Cluster+Rerank	0.560	0.811	0.413	0.56	0.5

B. Efficiency

We consider the following aspects of efficiency: the cost of index creation, the cost of query processing, and scalability. We focus on the three approaches to computing expertise since computing authority using the re-ranking method is much faster and takes much less space.

1) On Index Creation: Index creation has two major components: inverted list generation and list sorting. We also consider the index size, which indicates the space cost to store the index for the different methods. Table VII shows the time and space costs for the index creation for our approaches.

TABLE VII
TIME AND SPACE COST FOR INDEXING

Method	List Generation Time	List Sorting Time	Index Size
Profile	153 min	145 min	490 MB
Thread	148 min	435 min	502 + 40.2 MB
Cluster	142 min	0.4 min	48.8 + 0.9 MB

As mentioned previously, the list generation time is the same for the three approaches. The results in the table show clearly that the time differences for list generation are insignificant. However in practice, the list generation for the profile-based model is slightly more complicated, since it needs to combine the language models built on threads to create the users' profiles. So, the time for generating the inverted lists for the profile-based model would be slightly higher than those of the other two approaches.

For list sorting, the time complexity for each approach is $O(nm \log m)$, $O(nd \log d) + O(dm \log m)$, and $O(cm \log m)$, respectively, where m is the number of users, d is the number of threads, and c is the number of clusters. In our *BaseSet*, $n > d > m > c$. Hence the thread-based model performs the worst for sorting the inverted lists, while the cluster-based model is the most efficient.

As for the index size, the thread-based model and the cluster-based model both have two kinds of lists. For example, the thread-based model on *BaseSet* needs $502MB$ to store its *thread lists* and $40.2MB$ for the *thread user contribution lists*. In terms of the total space cost, the cluster-based model needs the least space, followed by the profile-based model, and ending with the thread-based model. It appears that the thread-based model uses the most space for indexing. However, for an existing QA system that already has an inverted index on threads, we only need to compute and store the *thread user contribution list* for our task. In such circumstance, the thread-based model will take the least space.

2) On Query Processing: In this experiment, we evaluate the query efficiency by comparing the average search time needed to retrieve the top-10 results. Additionally, for all three approaches, we also compare the models with and without use of the threshold algorithm. We want to show that the ones with the threshold algorithm can speed up the query processing. For the thread-based model, we only present the results of applying the threshold algorithm on the first stage.

Table VIII shows the response time for the top-10 search results. We can see that the threshold algorithm is capable of significantly speeding up the querying processing. Among the three models, the cluster-based model is the most efficient, and the thread-based model performs the worst. These results can be explained by the index sizes for different approaches, which affect the number of lists accessed and thus the computational

cost. Although the thread-based model seems to perform the worst, it finds both the experts for a new question and a list of relevant threads (containing similar questions to the new question). It is reasonable for a CQA system to search for similar question threads before finding experts, and a CQA system usually has index to support the search for relevant threads. Hence, the thread-based model may be easily integrated with question search by leveraging the threads obtained by question search to find relevant experts without incurring extra cost. Put differently, the thread-based model fits well with the infrastructure available in CQA systems.

TABLE VIII
TOP-10 SEARCH TIME OF DIFFERENT APPROACHES

Method	Top-10 search (second)
Profile	5.80
Profile+Threshold	1.68
Thread	11.87
Thread+Threshold	4.82
Cluster	0.71
Cluster+Threshold	0.34

3) On Scalability: We report experimental results for the five thread sets in Table I to describe the scalability of our approaches, i.e., *Set60K*, *Set120K*, *Set180K*, *Set240K*, and *Set300K*. Since there is no annotated test data for these data sets, we only show the efficiency by varying the data size. Figure 5 shows the results for the *Top-10 Search Time* when the number of threads is increased.

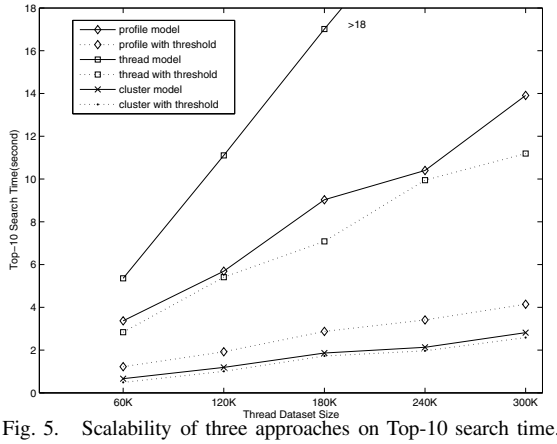

Fig. 5. Scalability of three approaches on Top-10 search time.

The figure shows that the performance of all approaches degrades when the data size increases, as they need to access more inverted lists to calculate the expertise of users. We can see that the models using the threshold algorithm are more stable than their counterparts without the threshold algorithm. Among them, the cluster-based model always performs the best. The figure shows that the cluster-based model without using the threshold algorithm also performs well. The reason might be that we have relatively few users in our data sets. We believe that the advantage of using the threshold algorithm might be more significant when we have a large number of users.

V. CONCLUSIONS

This paper describes a new approach to routing questions to the right users in online forums, or question answering portals. We present three approaches based on language models to represent the expertise of users based on their previous question answering activities. Experimental results on real data show that the proposed approaches can effectively find promising experts for new questions.

Several promising directions for future work exist. First, new threads are posted everyday in forums, creating a need to update the inverted indexes with new forum threads. It is easy to incrementally update the indexes of the thread-based model, while it appears to be nontrivial to update the indexes for the profile-based and cluster-based models. Second, it may be attractive to apply clustering techniques to generate fine grained clusters instead of using the clusters generated by sub-forums. Third, it would be interesting to investigate the usefulness of more complicated probabilistic models (e.g., [11]) for this task.

REFERENCES

[1] *Lucene Information Retrieval Library*. http://lucene.apache.org/.
[2] *TREC Enterprise Track*. http://trec.nist.gov/data/enterprise.html.
[3] K. Balog, L. Azzopardi, and M. de Rijke. Formal models for expert finding in enterprise corpora. In *Proc. of ACM SIGIR*, 43–50, 2006.
[4] K. Balog, T. Bogers, L. Azzopardi, M. de Rijke, and A. van den Bosch. Broad expertise retrieval in sparse data environments. In *Proc. of ACM SIGIR*, 551–558, 2007.
[5] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences*, 66(4).614–656, 2003.
[6] H. Fang and C. Zhai. Probabilistic models for expert finding. In *Proc. of ECIR*, 418–430, 2007.
[7] P. Jurczyk and E. Agichtein. Discovering authorities in question answer communities by using link analysis. In *Proc. of ACM CIKM*, 919–922, 2007.
[8] G. E. Littlepage and A. L. Mueller. Recognition and utilization of expertise in problem-solving groups: Expert characteristics and behavior. In *Group Dynamics: Theory, Research, and Practice*, 1.324–328, 1997.
[9] C. D. Manning, P. Raghavan, and Hinrich Schtze. Introduction to information retrieval. Cambridge University Press, 2008.
[10] D.W. McDonald and M. S. Ackerman. Expertise recommender: a flexible recommendation system and architecture. In *Proc. of ACM CSCW*, 231–240, 2000.
[11] D. Mimno and A. McCallum. Expertise modeling for matching papers with reviewers. In *Proc. of ACM SIGKDD*, 500–509, 2007.
[12] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.
[13] D. Petkova and W. B. Croft. Hierarchical language models for expert finding in enterprise corpora. In *Proc. of ICTAI*, 599–608, 2006.
[14] D. Petkova and W. B. Croft. Umass notebook 2006: Enterprise track. *Proc. of TREC-06*, 2007.
[15] J. M. Ponte and W. B. Croft. A language modeling approach to information retrieval. In *Proc. of ACM SIGIR*, 275–281, 1998.
[16] C. Shah and W. B. Croft. Evaluating high accuracy retrieval techniques. In *Proc. of ACM SIGIR*, 2–9, 2004.
[17] I. Soboroff, A. P. de Vries, and N. Craswell. Overview of the TREC 2006 Enterprise Track. *TREC 2006 Working Notes*, 2006.
[18] E. Voorhees and D. Tice. The TREC-8 question answering track evaluation. In *Proc. of TREC*, 83–105, 1999.
[19] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to information retrieval. *Journal of ACM TOIS*, 22(2).179–214, 2004.
[20] J. Zhang, M. S. Ackerman and L. Adamic. Expertise networks in online communities: structure and algorithms. In *Proc. of WWW*, 221–230, 2007.