Accelerating Instant Question Search with Database Techniques

Takeharu Eda Toshio Uchiyama Katsuji Bessho
Norifumi Katafuchi Alice Chen Ryoji Kataoka

NTT Cyber Solutions Laboratories, NTT Corporation
1-1 Hikari-no-oka, Yokosuka-shi, Kanagawa 239-0847 Japan
{eda.takeharu, uchiyama.toshio, bessho.katsuji,katafuchi.norifumi, alice.chen, kataoka.ryoji}@lab.ntt.co.jp

ABSTRACT

Distributed question answering services, like Yahoo Answer [1] and Aardvark [2], are known to be useful for end users and have also opened up numerous topics ranging in many research fields. In this paper, we propose a user-support tool for composing questions in such services. Our system incrementally recommends similar questions while users are typing their question in a sentence, which gives the users opportunities to know that there are similar questions that have already been solved. A question database is semantically analyzed and searched in the semantic space by boosting the performance of similarity searches with database techniques such as server/client caching and LSH (Locality Sensitive Hashing). The more text the user enters, the more similar the recommendations will become to the ultimately desired question. This unconscious *editing-as-a-sequence-of-searches* approach helps users to form their question incrementally through interactive supplementary information. Not only askers nor repliers, but also service providers have advantages such as that the knowledge of the service will be autonomously refined by avoiding for novice users to repeat questions which have been already solved.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Performance

Keywords

Question Authoring, Implementation, LSI, LSH

1. INTRODUCTION

Nowadays, distributed question answering (QA) services are ubiquitous; a user can casually ask any type of question in hope of receiving an expert answer. Even casual information surfers are well satisfied because QA services can hold a huge number of solved problems (QA database) relating to possibly anything in life [1].

From the viewpoint of knowledge sharing, there is a trilemma in QA databases as shown in Fig. 1. In order to compose readable and reasonable sentences, the asker needs to do several surveys through the web or the QA database before submitting a question. When repliers see repeated questions, they will insert several URLs linking previous resolved questions, and may occasionally request askers to clarify their questions in the case of ambiguous questions. As a result, surfers and the service provider may consider the QA database redundant and find it tedious to discover the knowledge they want.

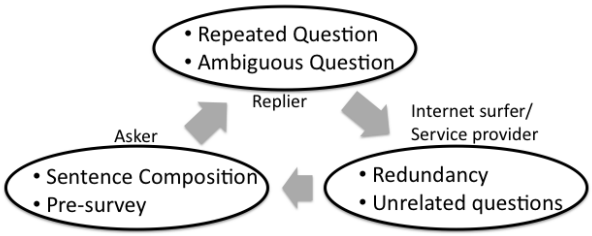

Figure 1: The Trilemma in QA Services .

Thus, we need some method for users, especially beginners, to avoid asking ambiguous questions or repeated questions that are likely to receive replies containing only links to other questions. Our approach is to assist users to edit their questions by recommending similar solved questions as they input their sentences. If the user found the almost same question she is going to ask, she would be satisfied without asking any question. If she could not find the question in the recommended questions, she will ask a new question in the service, which will make the QA service the richer knowledge database.

Retrieval of similar questions in our tool is achieved by combining VSM (Vector Space Model) and LSI (Latent Semantic Indexing). To improve the efficiency of the process, we also employ LSH (Locality Sensitive Hashing) in addition to traditional inverted lists. Question database is statistically analyzed and mapped into a high-dimensional vector space where similarity search is accelerated by indexing with LSH. Here, we have achieved a new interaction: incremental *concept* search, where semantically similar questions are immediately returned while users edit their question. We present several representative use cases in our demo.

2. PROPOSED SYSTEM

Fig. 2 shows the overall processing flow and the system architecture. Our system creates two indexes; inverted lists and LSH index, and the query processor combines both of the returned results. Such combination is a common technique for information

[1] http://answers.yahoo.com/
[2] http://vark.com/

WWW 2011, March 28–April 1, 2011, Hyderabad, India.
ACM 978-1-4503-0637-9/11/03.

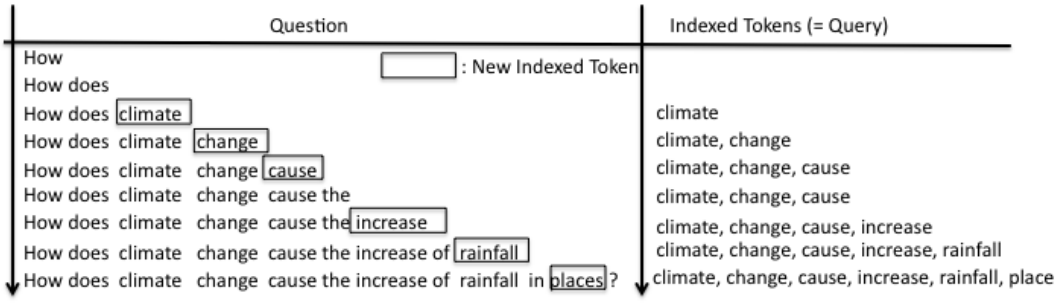

Figure 3: An Example Question. New indexed tokens are added while users edit question.

retrieval [3]. The reason why we chose this technique is that we can gain benefits from both merits, namely the high precision for small queries in VSM and better recall in LSI. First, we analyze the question database using Wikipedia or Iwanami (Japanese) dictionary and map words into a semantic space, a vector space with dimensionality greater than one hundred (Concept Database). Then, question vectors in the concept database are indexed with LSH, and similarity search based on their semantics can be processed very efficiently. As the user types in a question, it is segmented into

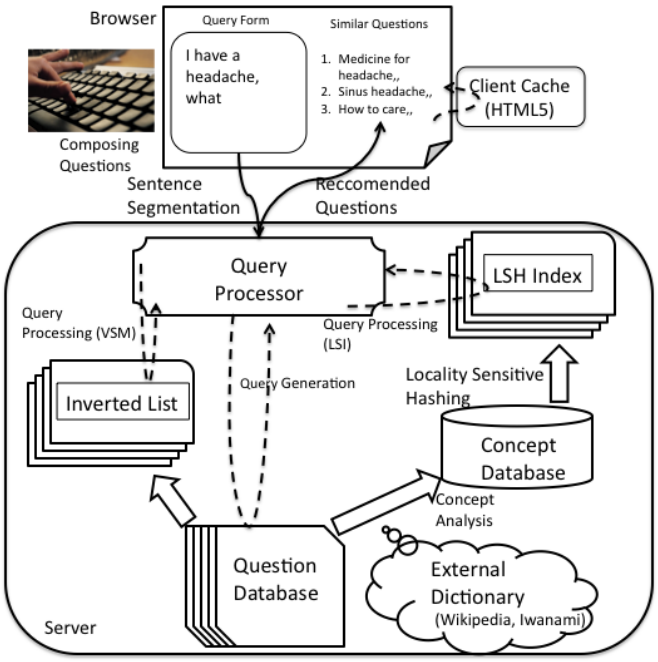

Figure 2: System Architecture.

tokens, and only indexed tokens are used as to query the semantic space. Finally, we calculate the centroid vector of the indexed tokens and search the LSH index for similar questions.

2.1 Incremental Concept Search

In incremental keyword search (or instant keyword search), one or more possible matches of documents containing the input keyword are presented to the user immediately. In incremental *concept* search, documents are presented immediately to the user. But they do not necessarily have the matches of documents containing the input keyword. They are similar to the input keywords in the semantic space and may contain similar words to the input keyword.

When the user types her sentence, the sentence is segmented into tokens. From the set of tokens, only indexed tokens are used as query for searching similar questions. As users add more tokens, more indexed tokens are included and the query changes (Fig. 3). As a result, the user gets a different set of similar questions.

Since conceptually-similar questions are returned immediately while users are adding/deleting tokens, they do not need to have the complete sentence of the question in mind during the search. Instead, they can edit and improve their sentences according to the recommended questions available to them. Once a desired question is spotted, the users can stop composing their question and access the targeted link directly.

2.2 Concept-based Question Search

To locate similar questions, we employed a concept-based search technique, a variant of latent semantic indexing technique [2]. Let X be p×q co-occurrence matrix between words and questions. X is decomposed by SVD as follows.

$$X = U\Sigma V^T$$

V^T is the inverse matrix of V . $r = rank X \leq \min(p, q)$, $U^T U = V^T V = I$ holds. Let $\Sigma = (\sigma_{ij}), \sigma_{ii} \geq \sigma_{jj} \geq 0 (1 \leq i \leq j \leq r), \sigma_{ij} = 0 (i \neq j)$. We call $\sigma_{ii (1 \leq i \leq r)}$ singular values. Given $r' (1 \leq r' \leq r)$, r' first columns from U, r' first columns from V^T, and r' first columns and rows from Σ, the following equation holds.

$$X' = U' \Sigma' V'^T$$

We call the normalized column vectors of U' *concept word vectors*.

In this model, we consider a question Q as the centroid vector V_Q of the concept word vectors the question covers.

$$V_Q = \frac{\Sigma_{w_i \in Q} w_i}{K_Q}$$

Here, K_Q is a normalized constant. All questions are represented as multidimensional vectors and indexed with LSH.

Intuitively, our model measures each question with its indexed tokens, which are statistically analyzed in the concept word vector space. The reason why we employed this semantic smoothing approach and rather than ordinary keyword search only is that this rich semantic model has the power to capture the latent semantics behind the question database. In other words, this method can locate semantically similar questions that do not necessarily have the exact keywords entered by users, but that share semantically similar keywords.

2.3 Locality Sensitive Hashing

Locality Sensitive Hashing (LSH) is an algorithm for solving approximate near neighbor search in high dimensional vector spaces.

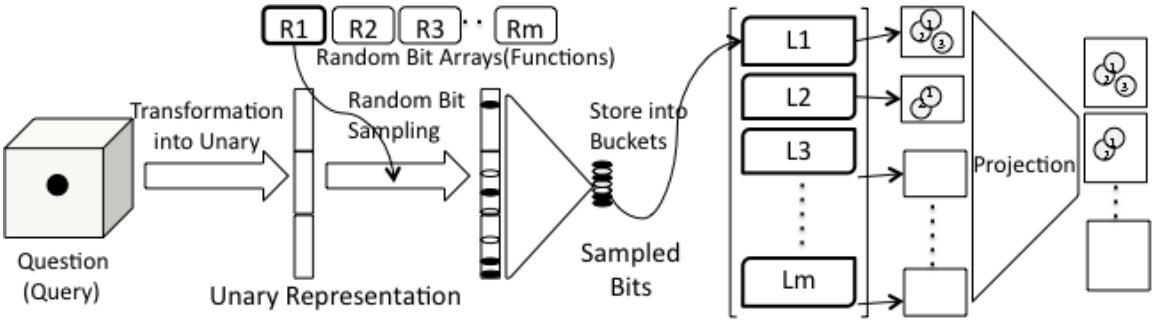

Figure 4: Locality Sensitive Hashing.

The key idea is to hash the points using several hash functions to ensure that for each function, the probability of collision is much higher for objects which are close to each other than for those that are far apart.

We employed an algorithm that uses unary representation of vectors and approximates l_1 distance [5]. Fig. 4 shows how LSH is used to index question vectors. Prior to indexing, random bit arrays, are created, each of which has the same number of randomly chosen bit positions for sampling. A question vector is concatenated onto a string and the string is virtually transformed into unary representation. The bit values, which are specified by random bit arrays, are concatenated into sampled bits and stored in a bucket. When the number of buckets becomes too large, the buckets are projected into another set of buckets. Given two questions, the more frequent the sampled bits are stored in the same bucket, the more similar the two vectors are.

3. IMPLEMENTATION TECHNIQUES

Although both concept search and LSH are well-established techniques, we implemented them into one system in order to improve the response time of incremental concept search.

3.1 Detection of Index Tokens and Server/Client Caching

In our scenario, users incrementally enter words in the search form. If a new word is an indexed token, the centroid vector of the question is updated and a new search is initiated. On the other hand, if a new word is not an indexed token, the centroid vector is not changed and the search result set does not need to be updated. Thus, we cache the previous indexed tokens and detect any changes while users are entering their question. This simple server caching mechanism has a significant impact on the response time of incremental concept search due to the fact that many tokens in a question are likely to be non-indexed and can be omitted.

As for the performance concern for longer queries of VSM (Fig. 5), we implemented a novel client caching mechanism in a browser using client side database storage [3]. In our system, the previous search result is cached in local storage. The local cache is filtered with a newly inserted keyword and reranked based on the approximate similarity to the query before sending the new request to server. Cache refresh is triggered by the similarity score between the current query and the documents in local cache. Thus, users do not need to refresh the local cache manually .

3.2 Grouping Bit Positions

In the original LSH, random positions are chosen randomly among each random bit array as shown in the lower part of Fig. 7. This makes sampling bit values from unary representation costly since the positions may cross among the arrays, thus we need multiple scans of unary representation.

To improve the situation, we implemented the random bit arrays by grouping them in the order of positions as shown in the upper part of Fig. 7. At first, all of required positions are randomly chosen, then we sort the positions and group them. By doing so, we can get bit values with just a single scan of the unary representation. This bit array design has performance impact not only on index construction (Fig. 6) but also on query processing. Furthermore, since the random array positions do not cross each other, we can benefit from the parallel processing of the random array.

While this technique provides no theoretical guarantee, we observed that the precision of similarity search is not degraded for the *dense* vector space in LSI compared to naive LSH.

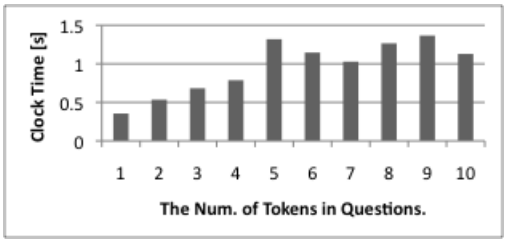

Figure 5: Average query processing time of VSM at server-side. For each Num. of tokens, ten queries are chosen randomly and the average processing time is reported. The performance of inverted lists degrades when the number of tokens in questions increases.

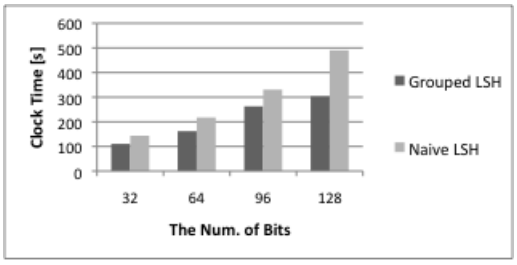

Figure 6: Clock time for constructing LSH indices in the cases of the Num. of hashes is 8 for a Japanese QA database.

4. USER EXPERIENCES

The performance achieved by our database implementation technique is significant. From a question database holding more than 2

[3] http://www.w3.org/TR/webdatabase/

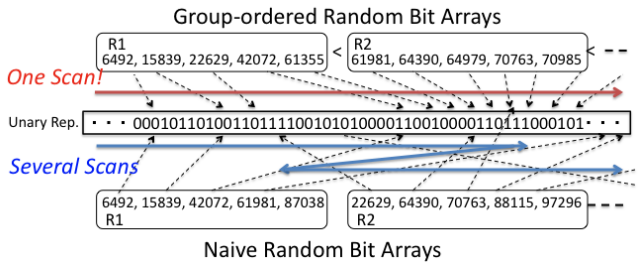

Figure 7: Grouping Bit Positions.

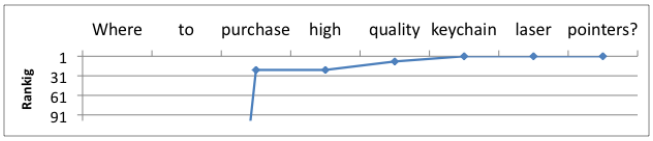

Figure 8: Topic Convergence: How the rank of the original question is changed while a user composes the question as a query? The more the user inserts the tokens, the higher the original document is ranked.

million entries, similar questions can be retrieved in less than 0.1[s] on average.

4.1 Demo Scenario

We've prepared the representative three use cases for our demo.

1. *Existing question search*
2. *New question search*
3. *Local cache performance comparison*

In the existing question search, users search questions existing in the demo system. While a user enters her own question, she will see that the range of topics covered in the pool of recommended questions gradually changes and feel as if the results are converging to the targeted question(Fig. 8) . In the new question search, users can try to search the question in her mind. In the local cache performance comparison, users can compare the recommended questions and the response times of retrieving similar questions in both cases of server side query processing and local filtering(Fig. 9).

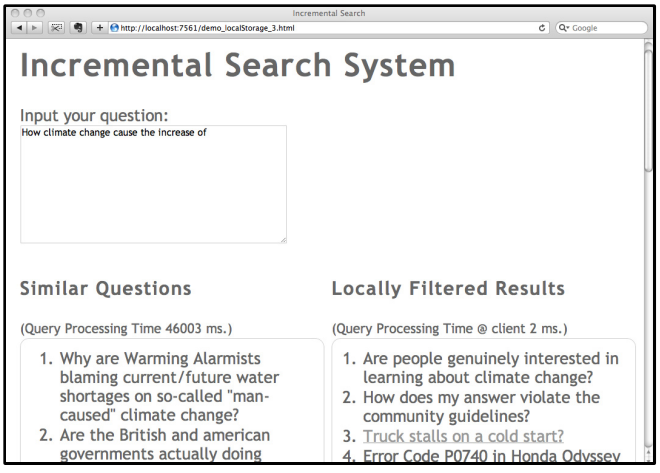

Figure 9: An Interface of Our Demo System.

At this point, we have utilized a Japanese question answering dataset (Oshiete-goo[4]) composed of more than two million questions and an English question answering dataset composed of more than one hundred thousands questions which are still being crawled now. We will be able to offer attendees the chance to try our incremental concept search system in both English and Japanese.

5. RELATED WORK

There is some research on composing questions through conceptual authoring but with a rather static setting [6]. The development of Web 2.0 popularized Ajax-like *dynamic* interaction between users and data [4]. This paper shows that indexing algorithms for high-dimensional vector space [5] enable us to enhance Ajax-like interaction in such a *dense* vectors space like the one in LSI, which is considered to be difficult to apply with inverted list techniques [3] .

Recently, Ji et al. [7] proposed the interactive fuzzy keyword search, where auto-complete keyword search is enhanced to support approximate keyword matching. They developed interactive tool by explored efficient indexing algorithms. However, the matching is approximate, as opposed to the semantic similarity matching we exploited in this paper, both of which are complementary and should be combined together for future work.

6. CONCLUSIONS

This paper proposes a user support tool that uses natural language processing and database engineering to present recommendations (solved questions) to users for composing questions in distributed question answering services. It incrementally recommends similar questions while users are entering their question. The question database is semantically analyzed and searched in the semantic space, and the performance is enhanced by utilizing LSH. Our system provides a new editing environment by a sequence of similarity searches and offers users opportunities to improve the sentences forming their questions.

Future directions include large-scale user evaluations in actuality of their realistic settings. Since the semantic smoothing model used in this paper is fully static, it is a more practical approach to employ recently proposed incremental model [8] for a wholly dynamic and autonomous incremental concept search system.

7. REFERENCES

[1] Lada A. Adamic, Jun Zhang, Eytan Bakshy, and Mark S. Ackerman. Knowledge Sharing and Yahoo Answers: Everyone Knows Something. In *Proc. WWW*, pages 665–674, 2008.

[2] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1999.

[3] Susan T. Dumais. Latent semantic indexing(lsi): Trec-3 report. In *Proc. Text REtrieval Conference(TREC-3)*, 1995.

[4] Jesse James Garrett. Ajax: A new approach to web applications. http://www.adaptivepath.com/publications/essays/archi-ves/000385.php, 2005.

[5] Aristides Gionis, Piotr Indyk, and Rajeev Motowani. Similarity Search in High Dimensions via Hashing. In *Proc. VLDB*, pages 518–529, 1999.

[6] Catalina Hallett, Richard Power, and Donia Scott. Composing Questions through Conceptual Authoring. *Computational Linguistics*, 33(1):105–133, 2007.

[7] Shengyue Ji, Guoliang Li, Chen Li, and Jianhua Feng. Efficient Interactive Fuzzy Keyword Search. In *Proc. WWW*, pages 371–380, 2009.

[8] Hu Wu, Dong Zhang, Yongji Wang, and Xiang Cheng. Incremental Probabilistic Latent Semantic Analysis for Automatic Question Recommendation. In *Proc. Recommender Systems*, pages 99–106, 2008.

[4]http://oshiete.goo.ne.jp/