Query Suggestions for Textual Problem Solution Repositories

Deepak P.[1], Sutanu Chakraborti[2], and Deepak Khemani[2]

[1] IBM Research - India, Bangalore, India
[2] Indian Institute of Technology Madras, India
deepak.s.p@in.ibm.com, {sutanuc,khemani}@iitm.ac.in

Abstract. Textual problem-solution repositories are available today in various forms, most commonly as problem-solution pairs from community question answering systems. Modern search engines that operate on the web can suggest possible completions in real-time for users as they type in queries. We study the problem of generating intelligent query suggestions for users of customized search systems that enable querying over problem-solution repositories. Due to the small scale and specialized nature of such systems, we often do not have the luxury of depending on query logs for finding query suggestions. We propose a retrieval model for generating query suggestions for search on a set of problem solution pairs. We harness the problem solution partition inherent in such repositories to improve upon traditional query suggestion mechanisms designed for systems that search over general textual corpora. We evaluate our technique over real problem-solution datasets and illustrate that our technique provides large and statistically significant improvements over the state-of-the-art technique in query suggestion.

1 Introduction

Query suggestions are a common feature in most modern web search engines (e.g., Google Instant[1]). Search engines that implement query suggestions assist users by providing a list of possible query completions as and when the user starts to type in a query in the search box. Recent studies report compelling empirical evidence in favor of usage of query suggestions; Feuer et. al.[1] report 30% uptake of query suggestions based on a comparative analysis of massive query logs before and after the query suggestion feature was included. Google reports that its query suggestion, Google Instant, enabled an average reduction of 2-5 seconds of effort in entering search queries. Query suggestions have now become a standard feature in most search engines, e.g., Bing, Baidu and Yandex[2].

Most of the state-of-the-art techniques for query suggestions exploit query log data (e.g.,[2,3,4]) as the primary resource. Plentiful availability of historical search information such as query logs is a workable assumption for query suggestion in

[1] http://www.google.com/insidesearch/features/instant/about.html
[2] http://en.wikipedia.org/wiki/List_of_search_engines

web-scale information retrieval engines since enough redundancy is ensured by the presence of a large number of users and search queries. In contrast, customized search engines such as those that operate on specialized domains, viz. intranet search, desktop search, personal email search and people search, do not have the luxury of depending on query logs due to their small user base. Query logs are unavailable when search systems are newly deployed, and even after long term usage, query logs do not accumulate fast enough due to the small user base; desktop search and personal email search are extreme cases where the user base is a single person. Advanced features such as query suggestions cannot wait for user logs to accumulate since the presence of such features often is key to adoption of such search systems among the limited user base. Towards enabling query suggestions in systems where query logs are scarce or absent, a recent work proposes a technique that harvests phrases from the text corpus and scores them based on factors such as the correlation with the user-entered query[5].

In this paper, we address the problem of *generating query suggestions for search systems that operate on problem solution repositories*[3] *where query logs are unavailable or scarce*. Problem solution repositories are available from a variety of sources, the most common web-source being community-driven question answering sites (CQA) such as Yahoo! Answers[4]. Textual problem solution pairs are readily available within service delivery organizations such as contact centers where problem tickets comprise of a textual description of the problem, and the solution steps as recorded by the human agent. Experience and incident reports may be segmented to the component problem-solution parts with high accuracy [6]. Since QA corpora comprise valuable historical knowledge pertaining to the business, reusing them is crucial to organizations that would want to avoid laborious re-discovery of previously used and known solutions and thus progressively improve response times for problem solving. Deploying search systems are an intuitive way to aid easy reuse of such accumulated organizational knowledge.

1.1 Why Specialized Query Suggestions for QA Systems?

In the presence of query suggestion techniques for general corpora, we motivate the need for specialized suggesters for QA systems by a three-fold argument:

1. Problem-Solution Partition: It has been shown that retrieval in QA system may be improved by upto 25%[7] over traditional document retrieval models by exploiting the problem-solution partition. Similarly, we expect and later illustrate that QA-aware query suggestion can outperform generic methods.

2. Usage of a QA Query System: A user typically queries an QA system when she only has a few keywords-worth knowledge of the problem to be solved. The query suggestion engine, intuitively, is expected to guide her to modify the query to an extent where relevant solutions are available in the QA repository. Consider a query *'android emulator'*; a frequency-driven suggester may pick *'android emulator download'* due to having a high frequency compared to *'download*

[3] We use Problem solution repositories and QA repositories interchangeably.

[4] `http://answers.yahoo.com/`

android emulator for windows' or *'android emulator linux'*. However, since it is unlikely that there are android emulators that are generic across platforms, latter phrases may be better suggestions to avoid repetetive query refinements. Such **underspecification** may be avoided if the suggester is QA aware; *'android emulator download'* may be de-prioritized by a QA-aware suggester that identifies that solution parts for *windows* and *linux* sub-queries for android emulators lead to significantly different solutions. Further, generic suggesters may **overspecialize** queries such as *'jellybean'* to *'jellybean update'* if numerous users have inquired about the latter phrase, even if no satisfactory set of answers are present in the corpus. Thus, a QA-aware suggester can ease the user's task to easily arriving at the desired level of specialization for the corpus.

3. Avoiding 'Useless' Suggestions: Certain phrases may be common in a corpus due to reasons other than the presence of problems pertaining to them; for example, *'phone call'* may be common due to reasons as varied as the customer referring to a previous phone call in a follow-up request and another document talking about a very different problem being solved after a phone call with an expert and so on. Generic suggestion engines may bring up *'phone call'* as a suggestion to an incomplete query such as *'phone'* or *'call'*. QA aware engines can identify from the obviously high lexical scatter of documents containing the phrase *'phone call'* that no core issues pertaining to calling on a phone have been solved in the corpus and can avoid suggesting such phrases. Avoiding phrases that pertain to problems not solved by documents in the repository is key to minimize time wastage and dissatisfaction for the user.

Despite such arguments, frequent phrases are advantageous since they lead to more documents for the user. We enrich frequency-driven scoring by blending QA-information using concepts from Textual CBR[5] *and language modeling.*

2 Related Work

Since our problem spans areas such as processing of QA repositories, CBR and IR query suggestions, we present a categorized summary of related work.

Interactive Query Refinement (IQE): IQE[8] starts with a query that represents a fully-specified information need and attempts to find meaningful specializations, most commonly, as words to augment the query. For example, *'world cup'* could be an initial query whose refinement suggestions by an IQE system may include terms such as *football* and *2010*. On the other hand, query suggestion systems take *incomplete* queries as input and produce *completions* to it so that the user may save typing time and may be exposed to various ways of completing the incompletely specified information need. Thus, *'world'* and *'world c'* could form input to the query suggestion system whereas they being *incomplete* information needs, would not be meant to be dealt with by IQE systems. This difference has led to solutions to these two problems evolve differently.

Query Suggestions using Search History: Query logs have been heavily exploited for query suggestion since modifications of queries in the same search session

[5] `http://en.wikipedia.org/wiki/Case-based_reasoning`

provide valuable reusable information. Modifications previously made by users may be used [9] to derive query suggestions. Clickthrough data and session information have been used to capture the *context* of a user query as a sequence of concepts that is further used to condition the query suggestion engine to produce more context-aware query suggestions [2]. The query flow graph approach [10] involves modeling past user queries that pertain to the same search need as connected nodes in a graph; short random walks starting from the available incomplete query are used to generate query recommendations. [11] propose to suggest similar queries based on historical user landing page correlations.

Query Suggestions for Specialized Search Systems: Specialized small-scale search engines, as observed earlier, do not have plentiful search histories and hence have to depend on term statistics in the corpus to score phrases. The CompleteSearch system [12] generates completions based on the number of search results for each possible completion; thus, for a query *'world c'*, *'world cup'* would be ranked higher than *'world clock'* if the former produces larger number of results. CompleteSearch focuses on building indexes so that the calculation of the number of search results may be done very fast at query time. Sumit et. al. [5] improve upon such a simplistic frequency based scoring by using a probabilistic scoring where a candidate query suggestion is scored based on (1) the correlation to the query estimated using the number of documents they co-occur in the corpus and (2) the IDF of the terms in the candidate. Other specialized query suggestion systems on the web work by restricting the problem scope to a specific result type such as *entity names*; examples include Facebook[6] where the result domain comprises of only people names and group names. Apart from the task being simpler, the search results then may be ranked based on features such as proximity of the candidate entity to the query issuer in the social graph.

Retrieval and Suggestions for QA Repositories: Query suggestions for QA repositories are meant to augment existing search systems on QA collections. QA collections, being comprised of documents that have a two-part structure, that of a problem and a solution, pose significant challenges to retrieval due to the *lexical chasm*, the lexical gap between the two parts. Since the user posed query may be expected to be behaviorally similar to the lexical behavior of questions in the QA repository, the query needs to be conceptually 'expanded' to include correlated solution words if relevant QA pairs may be retrieved using traditional IR techniques. IBM Model 1 [13] has been used in bridging the lexical chasm [14,7] for retrieval tasks in QA collections. Since [7], *Question Suggestion*, the problem of finding similar questions to a given *question (problem)* in QA collections (and hence, different from our problem of *query suggestions* which is meant to complete a partial information need) has centered around the usage of translation model variants such as topic-enhanced [15] and phrase level models [16]. Similar to observations in the retrieval task where differential treatment of question and answer parts leads to significant gains, we will illustrate that such differentiation is useful for the query suggestion problem too.

[6] `http://www.facebook.com`

3 Problem Definition

Consider a retrieval system that operates on a corpus of QA documents, $\mathcal{D} = \{d_1 = (q_1, a_1), \ldots, d_n = (q_n, a_n)\}$ where d_i represents a document comprising of the question q_i and answer a_i. Let $\mathcal{P}$ be a set of n-grams of length up to x words[7] (i.e., phrases) $\mathcal{P} = \{p_1, p_2, \ldots, p_m\}$ from among documents in $\mathcal{D}$.

Now, for a user who has so far typed in an incomplete query Q, we would like to suggest a set of k phrases from among those in $\mathcal{P}$ that are most likely completions of the incomplete query Q. In particular, we develop a scoring function, $\mathcal{S}(p, Q, \mathcal{D})$ that scores each phrase $p \in \mathcal{P}$ wrt its relevance as a completion of Q for a retrieval system that operates on $\mathcal{D}$. Given such a scoring function, the top-k suggestions for Q are the top-k highest scoring phrases from $\mathcal{P}$.

4 QuerySuggest-QA: The Proposed Technique

We design our scoring function, $\mathcal{S}(p, Q, \mathcal{D})$, as being comprised of two parts that are composed using a simple product formulation:

- $P_f(p, \mathcal{D})$: This roughly models relative frequency of the phrase p in the corpus as against other phrases in $\mathcal{P}$ and is independent of Q and is the *frequency driven component* of our scoring function.
- $P_{\mathcal{D}}(p|Q)$: This part models the probability of choosing the phrase p for an incomplete query Q for the corpus $\mathcal{D}$ and is the *QA-aware component* that differentiates our approach from generic query suggesters.

$$\mathcal{S}(p, Q, \mathcal{D}) = P_f(p, \mathcal{D}) \times P_{\mathcal{D}}(p|Q)$$

Since we expect user queries to be more close to the behavior of the questions in the corpus, we apply a relative weighting between the question and answer part in the corpus using a parameter λ. The weighting for the question and answer parts would then be λ and $(1 - \lambda)$ respectively. We will use λ in modeling both parts of our scoring function, thus making *even the first part slightly QA-aware.*

4.1 Relative Phrase Frequency

In this criterion, we prefer phrases that are more frequent in $\mathcal{D}$ since those would intuitively satisfy more users (assuming users' problems behave similarly to those in $\mathcal{D}$). Such considerations are heavily used in previous work (e.g.,[12,5]). A simple formulation would be to normalize the phrase score by the sum of the frequencies across phrases in $\mathcal{P}$ to arrive at:

$$P_f(p, \mathcal{D}) = \frac{f(p, \mathcal{D})}{\sum_{p' \in \mathcal{P}} f(p', \mathcal{D})}$$

[7] A word n-gram is a phrase comprising of n contiguous words in text; for example, *world cup* is a word 2-gram and *US of A* is a word 3-gram. We use x=5.

Since such a frequency based measure would be biased in favor of shorter phrases [17], we use $f(.,.)$ to denote normalized frequencies as proposed in [5]:

$$f(p, \mathcal{D}) = \frac{\#times(p, \mathcal{D})}{log\{avg\{\#times(p', \mathcal{D}) | p \in \mathcal{P} \wedge size(p') = size(p)\}\}}$$

where $size(p)$ denotes the number of words in p. Further, we model $\#times(.,.)$ as the sum of the weighted frequency using the weighting parameter λ.

$$\#times(p, \mathcal{D}) = \lambda\ \#times(p, \{q_1, \ldots, q_n\}) + (1 - \lambda)\ \#times(p, \{a_1, \ldots, a_n\})$$

where $\#times(p, S)$ denotes the number of entries in S containing p.

4.2 Modeling QA-Aware Phrase Probability

In modeling this criterion, we exploit the correlations between questions and answers in $\mathcal{D}$. First, we use Bayes theorem to rewrite the expression as:

$$P_{\mathcal{D}}(p|Q) = \frac{P_{\mathcal{D}}(p) \times P_{\mathcal{D}}(Q|p)}{P_{\mathcal{D}}(Q)} \equiv P_{\mathcal{D}}(p) \times P_{\mathcal{D}}(Q|p)$$

Avoiding $P_{\mathcal{D}}(Q)$ (denominator) is kosher since all phrases are being evaluated wrt Q. We now marginalize the $P_{\mathcal{D}}(Q|p)$ over the documents in $\mathcal{D}$:

$$P_{\mathcal{D}}(p) \times P_{\mathcal{D}}(Q|p) = P_{\mathcal{D}}(p) \times \sum_{d \in \mathcal{D}} P_{\mathcal{D}}(Q, d|p)$$

Further,

$$P_{\mathcal{D}}(p) \times \sum_{d \in \mathcal{D}} P_{\mathcal{D}}(Q, d|p) = P_{\mathcal{D}}(p) \times \sum_{d \in \mathcal{D}} \frac{P_{\mathcal{D}}(Q, d, p)}{P_{\mathcal{D}}(p)}$$

$$= \cancel{P_{\mathcal{D}}(p)} \times \sum_{d \in \mathcal{D}} \frac{P_{\mathcal{D}}(Q, d, p)}{\cancel{P_{\mathcal{D}}(p)}} \qquad \text{(cancelling out)}$$

$$= \sum_{d \in \mathcal{D}} P_{\mathcal{D}}(Q|d, p) \times P_{\mathcal{D}}(d) \times P_{\mathcal{D}}(p|d) \qquad \text{(by chain rule)}$$

$$= \sum_{d \in \mathcal{D}} \frac{P_{\mathcal{D}}(Q|p) \times P_{\mathcal{D}}(d|Q, p)}{P_{\mathcal{D}}(d|p)} \times P_{\mathcal{D}}(d) \times P_{\mathcal{D}}(p|d) \qquad \text{(by Bayes rule)}$$

Most query suggestion systems [12,5] consider only the subset of phrases that contain the query, as candidate suggestions. We too propose to restrict our search to those phrases that contain the query[8]; under this assumption, $P_{\mathcal{D}}(Q|p)$ evaluates to 1.0 since the presence of the phrase p implies the presence of its sub-part Q. $P_{\mathcal{D}}(d|Q, p)$ evaluates to $P_{\mathcal{D}}(d|p)$ due to a similar reason. Thus,

[8] E.g., *'world cup'.* contains *'world'* and *'world c'.*

$$= \sum_{d \in \mathcal{D}} \frac{\cancelto{1.0}{P_{\mathcal{D}}(Q|p)} \times \cancel{P_{\mathcal{D}}(d|Q,p)}}{\cancel{P_{\mathcal{D}}(d|p)}} \times P_{\mathcal{D}}(d) \times P_{\mathcal{D}}(p|d)$$

$$= \sum_{d \in \mathcal{D}} \underbrace{P_{\mathcal{D}}(d)}_{\text{document importance}} \times \underbrace{P_{\mathcal{D}}(p|d)}_{\text{phrase probability from document}}$$

Computing Document Importance: Based on the motivation from Section 1.1, we would like to suggest phrases such that there are *many problems pertaining to them* in $\mathcal{D}$ that have been *solved well.* Textual CBR relies on the reusability of similar solutions for similar problems [18]. Thus, good solutions to a given problem are likely to be partly or fully reused for similar problems; i.e., a good solution to a problem is likely to have many lexically similar solutions if many similar problems occur in the repository. Such considerations are not new and have been well explored in the area of case-based reasoning, where case-base alignment[9] measures [18] quantify such notions. At the document level, these translate to preferring a document $d = (q, a)$ if:

- *$\mathcal{D}$ contains many problems similar to q:* Unless there are many similar problems, there is not enough redundancy to assess how well solved the problem addressed by d is.
- *Problems in $\mathcal{D}$ similar to q have solutions similar to a:* Such a scenario enhances the confidence that d contains a *good* solution.

We quantify similarity as the lexical similarity between document parts, which may be calculated using popular text similarity metrics such as cosine similarity of tf.idf vectors[10]. For d, though we would like to have many similar problems, we consider only t most similar problems to avoid biasing our suggestions towards very common problem clusters. For each document d, we pick the top-t documents from $\mathcal{D}$, $\{d^1_{\mathcal{D}}, \ldots, d^t_{\mathcal{D}}\}$, based on similarity between question (problem) parts. Our importance metric now estimates the product of the question part similarity and the answer part similarity (since the desired condition is that both are high) for each such selected case and sums it up over the t cases to arrive at a single measure for d.

$$CA_{\mathcal{D}}(d) = \sum_{1 \le i \le t} S_q(d, d^i_{\mathcal{D}}) \times S_a(d, d^i_{\mathcal{D}})$$

where $S_q(d_i, d_j)$ and $S_a(d_i, d_j)$ denote the lexical similarity between the problem and solution parts of d_i and d_j respectively. We use $CA_{\mathcal{D}}(.)$ to denote this metric owing to being motivated by (though significantly different from) the local **C**ase

[9] Case, in CBR parlance, refers to a problem-solution pair. Case bases are collections of cases such as a QA repository. Case-base alignment measures quantify how well 'similar problems have similar solutions' assumption holds in a case base.

[10] `http://en.wikipedia.org/wiki/Cosine_similarity`

Alignment metric in [19]. $P_{\mathcal{D}}(d)$ is then computed as the fractional importance of the document d in the corpus:

$$P_{\mathcal{D}}(d) = \frac{CA_{\mathcal{D}}(d)}{\sum_{d' \in \mathcal{D}} CA_{\mathcal{D}}(d')}$$

Phrase Probability from Document: For this part, we use the maximum likelihood estimate of the phrase based on a unigram language model derived from the document. Such estimation is very commonly used in language-model based IR [20]; we weigh the question and answer parts using λ:

$$P_{\mathcal{D}}(p|d) = \lambda \prod_{w \in p} \frac{count(w,q)}{\sum_{w' \in q} count(w',q)} + (1-\lambda) \prod_{w \in p} \frac{count(w,a)}{\sum_{w' \in a} count(w',a)}$$

where $d = (q, a)$ and $count(w, s)$ denotes the frequency of w in s. ■

4.3 Combined Formula

Given the restriction to phrases containing Q, the combined function becomes:

$$\mathcal{S}(p, Q, \mathcal{D}) = \begin{cases} 0, & \text{if p does not contain Q} \\ P_f(p, \mathcal{D}) \times \sum_{d \in \mathcal{D}} (P_{\mathcal{D}}(d) \times P_{\mathcal{D}}(p|d)), & \text{otherwise} \end{cases}$$

Similar to previous work on query suggestions, we do not enforce that p contains words in Q in the same order as in the latter. We consider it OK if p contains all words in Q in any order. If Q contains an incomplete word (e.g., *'pre'* in Q=*'insurance pre'*), we consider those phrases that have a valid completion for the incomplete word, as well as occurences of other complete words in Q.

Complexity: In a static collection $\mathcal{D}$, for every phrase in $\mathcal{D}$, the term in the second case of the combined formula above may be pre-computed. Thus, upon a query, one just needs to find the subset of phrases that contain it, and rank them according to the pre-computed scores to pick the top-k. Without indexes, this may be done in $\mathcal{O}(|\mathcal{P}| + k\ log\ b)$; for each phrase, the check for inclusion of the query takes constant time (since queries and phrases are typically only a few words long, leading to only $10s$ of checks) resulting in identifying the b phrases containing the query, after which a partial sort to find the top-k phrases is performed. This may be speeded up if IR-style reverse indexes are employed. When the QA collection is dynamic, wherein documents may be included and excluded over a period of time, most of the computation would have to be moved to runtime making query-time operations linear in both $|\mathcal{D}|$ and $|\mathcal{P}|$.

5 Experimental Evaluation

5.1 Datasets and Evaluation Measures

Datasets: We use three datasets of two-part QA documents in our experiments, from among those collected for an IR task[11]. We use three QA domains therein,

[11] `http://irsi.res.in/fire/faq-retrieval/Resources.html`

Insurance (990 documents) containing answers to common questions in the insurance sector, *Career* (938 documents) comprising of career-related QA and the smaller *Sports* collection (357 documents) having sports related Q&A. We choose phrases from among noise-corrected queries in the query collection supplied along with the dataset; either the first 1-2 words alone, or the first 1-2 words along with a few characters of the next word (to form the incomplete word), are taken from these phrases to be used as queries. We collected 45 queries in total, 16 from *Career*, 17 from *Insurance* and 12 from *Sports.* The suggestions generated from our approach and the baseline were collected and given to 1-2 human labelers who labeled the query suggestions as either relevant or irrelevant (binary). Since domain knowledge variations among labelers could lead to falsely underestimating the relevance, we merge the labelings by labeling a suggestion as relevant whenever there is a conflict. We then use the labelings to estimate the performance of the techniques wrt various evaluation measures.

Evaluation Measures: Apart from the standard IR evaluation measures[12] such as Precision, NDCG, MAP, MRR averaged across queries per-dataset, we also use the *Success Rate* [5], the number of queries for which at least one relevant suggestion is retrieved. We use randomization tests [21] to assess statistical significance of the performance of our technique against the baseline method.

Being a QA system, the suggested query may be considered good if the retrieved documents contain problems that are *well solved* within the QA repository. This is desirable since the only solution contained within the document itself may be insufficient, and more solutions from across similar documents in the collection would reduce dissatisfied users. This is a common scenario in CBR systems, and statistical measures are used to quantify such considerations for textual CBR; we use the *tot* measure [22][13] that, for every problem, estimates the total usability of solutions across other documents in the collection. For each query suggestion for Q, we pick the set of documents that would be retrieved for it in a standard IR system, and evaluate the *tot* measure for each document and average across the set to get a single value for the phrase. This is then averaged across all phrase suggestions produced by a technique for Q to serve as a measure of the performance of the technique for Q; higher values of *tot* are desirable.

5.2 Experimental Results and Discussion

Comparative Study: We evaluate our approach against the state-of-the-art algorithm for query suggestion in the absence of query logs [5] over general text corpora; we refer to it as *BMM*, based on the initials of the authors. Unless otherwise mentioned, we use λ=0.9 (Ref. Sec 4) for our approach. For most measures, we compare the top-10 suggestions generated by both the techniques (as indicated by @10).

[12] `http://nlp.stanford.edu/IR-book/html/htmledition/evaluation-of-ranked-retrieval-results-1.html`

[13] We are unable to delve into details of the *tot* measure due to space constraints.

Table 1. Results of Evaluation (∘ & • denote statistical significance at $p < 0.05$ and $p < 0.01$ respectively)

	Precision@10		MAP@10		MRR		NDCG@10		SR@1	
Dataset	BMM	QS-QA	BMM	QS-QA	BMM	QS-QA	BMM	QS-QA	BMM	QS-QA
Career	0.538	**0.775**$^\bullet$	0.688	**0.895**$^\bullet$	0.820	**1.0**$^\bullet$	0.668	**0.900**$^\bullet$	0.75	**1.0**$^\bullet$
Insurance	0.559	**0.694**$^\circ$	0.803	**0.839**	0.961	**1.0**$^\bullet$	0.818	**0.841**	0.941	**1.0**$^\bullet$
Sports	0.508	**0.617**$^\circ$	0.719	**0.805**	0.892	**0.958**$^\bullet$	0.728	**0.814**	0.833	**0.917**$^\bullet$
All	0.538	**0.702**$^\bullet$	0.740	**0.850**$^\bullet$	0.892	**0.989**$^\bullet$	0.741	**0.855**$^\bullet$	0.844	**0.978**$^\bullet$

Success Rate: Since our method, QS-QA, is QA-aware, it is able to prioritize relevant suggestions by weighing well-solved questions higher. This reflects in the Success Rate (@1) listed in Table 1; across datasets, the first result in QS-QA is found to be relevant 97.8% of the time whereas the corresponding figure for BMM is 84.4%. Figure 1 plots the success rates at varying k (i.e., #suggestions) where QS-QA is seen to retrieve at least one relevant suggestion among the top-2 for each query thereby reaching a success rate of 100% at k=2. On the contrary, there are queries where the first relevant suggestion from BMM is at rank 8.

Performance on Classical IR Evaluation Metrics: Similar trends as for SR are observed for Precision, MAP, MRR and NDCG measures as illustrated in Table 1. QS-QA outperforms BMM by roughly 18% on an average across metrics and datasets, while achieving significantly higher gains on the Precision value; the latter indicates that the good performance of QS-QA extends further down the list (the other metrics prioritize performance for the top few results). Similar figures were observed for these metrics when computed @5 too.

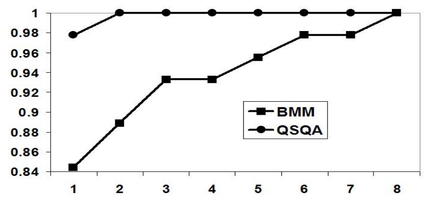

Fig. 1. *Success Rate* with varying k

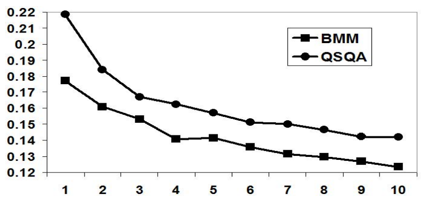

Fig. 2. *tot* with varying k

Performance on the *tot* Measure: The *tot* statistical measure computes the extent to which good solutions may be achieved using the suggested queries and does *NOT depend on the manual labelings* and hence is very complimentary to other metrics. We plot the *tot* measure at various cut-offs of the ranked lists in Figure 2. While QS-QA consistently ourperforms BMM on this measure too, gains over the latter range between 12% and 19% across values of k upto 10.

Sample Results: We present a few sample results for two queries from among the collection used. While these do not represent a comprehensive evaluation, these give a sense of the performance of our technique. For example, in the query *'card'*, BMM, driven by a frequentist approach, prefers to suggest *health card* related phrases. QS-QA is able to identify the relatively less frequent phrases such as *'id card'* and *'medicare card'*, which were found to be the central topic for certain queries in the collection. *'engineering de'* poses a more difficult query since the most common referent is that pertaining to *'engineering degree'*; even in this case, QS-QA is able to prioritize topical phrases higher.

Sensitivity to λ: We observed empirically that our technique is not very sensitive to λ. While the gains drop at either ends (0.0 & 1.0), reasonably consistent behavior was observed between 0.5 and 0.9. The performance was seen to drop gradually for values of λ less than 0.5 towards 0.0.

Table 2. Sample Results Comparison ($*$ is marked against suggestions deemed to be relevant by human labelers)

Domain: *Insurance*, Query: *'card'*		Domain: *Career*, Query: *'engineering de'*	
QuerySuggest-QA	BMM	QuerySuggest-QA	BMM
health card*	health card*	degree in engineering*	engineering in eee dept
credit card*	credit card*	engineering or degree	details marine engineering*
id card*	health card mentions	degree in an engineering	engineering would depend
medicare card*	card mentions the contact	degree in aeronautical engineering*	depend on performance in engineering
card which	health card function	details marine engineering*	degrees in computer engineering*

6 Conclusions and Future Work

In this paper, we considered the problem of generating query suggestions for IR over a QA collection. Typical IR systems over QA tend to be small scale making usage of query logs an infeasible propostion. The state-of-the-art in query suggestion in the absence of query logs [5] makes use of statistical properties of phrases. We proposed the usage of a QA-aware approach where phrases that are central to questions posed in the collection, as well as those from documents that are well-solved in the collection, are prioritized. Towards this, we integrated concepts from case-based reasoning and language modeling to form a ranking model for phrases. Our technique is seen to outperform the state-of-the-art BMM technique consistently on real-world datasets, and in most cases, statistically significantly. We also evaluated the utility of the query suggestions wrt leading the user towards documents where questions posed have been solved well within the collection. Though our technique is expected to score better on this evaluation (due to using related concepts in the suggestion generation process), we recorded significant gains of $12-18\%$ over BMM. In summary, we have explored and empirically established the utility of differential treatment of question and answer parts, and QA-aware statistical measures over QA collections, in generating query suggestions for a IR system over a QA collection.

In a follow-up, we are exploring directions to ensure diversity in query suggestions over QA repositories. We are also looking at developing a framework where additional information such as section information in text books, metadata in patents etc. may be plugged in to enhance query suggestion quality.

References

1. Feuer, A., Savev, S., Aslam, J.A.: Evaluation of phrasal query suggestions. In: Proceedings of the Sixteenth ACM Conference on Information and Knowledge Management, CIKM, pp. 841–848 (2007)
2. Cao, H., Jiang, D., Pei, J., He, Q., Liao, Z., Chen, E., Li, H.: Context-aware query suggestion by mining click-through and session data. In: KDD, pp. 875–883 (2008)
3. Ma, H., Yang, H., King, I., Lyu, M.R.: Learning latent semantic relations from clickthrough data for query suggestion. In: CIKM, pp. 709–718 (2008)
4. Song, Y., Wei He, L.: Optimal rare query suggestion with implicit user feedback. In: WWW, pp. 901–910 (2010)
5. Bhatia, S., Majumdar, D., Mitra, P.: Query suggestions in the absence of query logs. In: SIGIR, pp. 795–804 (2011)
6. Deepak, P., Visweswariah, K., Wiratunga, N., Sani, S.: Two-part segmentation of text documents. In: CIKM (2012)
7. Xue, X., Jeon, J., Croft, W.B.: Retrieval models for question and answer archives. In: SIGIR, pp. 475–482 (2008)
8. Fonseca, B.M., Golgher, P.B., Pôssas, B., Ribeiro-Neto, B.A., Ziviani, N.: Concept-based interactive query expansion. In: CIKM, pp. 696–703 (2005)
9. Jones, R., Rey, B., Madani, O., Greiner, W.: Generating query substitutions. In: WWW, pp. 387–396 (2006)
10. Boldi, P., Bonchi, F., Castillo, C., Donato, D., Vigna, S.: Query suggestions using query-flow graphs. In: Proceedings of the 2009 Workshop on Web Search Click Data, WSCD 2009, pp. 56–63 (2009)
11. Cucerzan, S., White, R.W.: Query suggestion based on user landing pages. In: Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR, pp. 875–876 (2007)
12. Bast, H., Weber, I.: Type less, find more: fast autocompletion search with a succinct index. In: SIGIR, pp. 364–371 (2006)
13. Brown, P.F., Cocke, J., Pietra, S.A.D., Pietra, V.J.D., Jelinek, F., Lafferty, J.D., Mercer, R.L., Roossin, P.S.: A statistical approach to machine translation. Comput. Linguist. 16(2), 79–85 (1990)
14. Jeon, J., Croft, W.B., Lee, J.H.: Finding similar questions in large question and answer archives. In: CIKM, pp. 84–90 (2005)
15. Zhou, T.C., Lin, C.Y., King, I., Lyu, M.R., Song, Y.I., Cao, Y.: Learning to suggest questions in online forums. In: AAAI (2011)
16. Zhou, G., Cai, L., Zhao, J., Liu, K.: Phrase-based translation model for question retrieval in community question answer archives. In: ACL, pp. 653–662 (2011)
17. Cui, H., Wen, J.R., Nie, J.Y., Ma, W.Y.: Probabilistic query expansion using query logs. In: WWW, pp. 325–332 (2002)
18. Raghunandan, M.A., Wiratunga, N., Chakraborti, S., Massie, S., Khemani, D.: Evaluation Measures for TCBR Systems. In: Althoff, K.-D., Bergmann, R., Minor, M., Hanft, A. (eds.) ECCBR 2008. LNCS (LNAI), vol. 5239, pp. 444–458. Springer, Heidelberg (2008)

19. Massie, S., Wiratunga, N., Craw, S., Donati, A., Vicari, E.: From Anomaly Reports to Cases. In: Weber, R.O., Richter, M.M. (eds.) ICCBR 2007. LNCS (LNAI), vol. 4626, pp. 359–373. Springer, Heidelberg (2007)
20. Liu, X., Croft, W.B.: Statistical language modeling for information retrieval. ARIST 39(1), 1–31 (2005)
21. Smucker, M.D., Allan, J., Carterette, B.: A comparison of statistical significance tests for information retrieval evaluation. In: CIKM, pp. 623–632 (2007)
22. Chakraborti, P.D., Khemani, S., More, D.: or better: on trade-offs in compacting textual problem solution repositories. In: CIKM, pp. 2321–2324 (2011)