Ranked Queries over Sources with Boolean Query Interfaces without Ranking Support

Vagelis Hristidis [#1], Yuheng Hu [#2], Panagiotis G. Ipeirotis [*3]

[#]*School of Computing and Information Sciences, Florida International University*
Miami, FL, USA
[1]vagelis@cis.fiu.edu
[2]yhu002@cis.fiu.edu

[*]*Department of Information, Operations, and Management Sciences; New York University*
New York, NY, USA
[3]panos@stern.nyu.edu

***Abstract*— Many online or local data sources provide powerful querying mechanisms but limited ranking capabilities. For instance, PubMed allows users to submit highly expressive Boolean keyword queries, but ranks the query results by date only. However, a user would typically prefer a ranking by relevance, measured by an Information Retrieval (IR) ranking function. The naive approach would be to submit a disjunctive query with all query keywords, retrieve the returned documents, and then re-rank them. Unfortunately, such an operation would be very expensive due to the large number of results returned by disjunctive queries. In this paper we present algorithms that return the top results for a query, ranked according to an IR-style ranking function, while operating on top of a source with a Boolean query interface with no ranking capabilities (or a ranking capability of no interest to the end user). The algorithms generate a series of conjunctive queries that return only documents that are candidates for being highly ranked according to a relevance metric. Our approach can also be applied to other settings where the ranking is monotonic on a set of factors (query keywords in IR) and the source query interface is a Boolean expression of these factors. Our comprehensive experimental evaluation on the PubMed database and TREC dataset show that we achieve order of magnitude improvement compared to the current baseline approaches.**

I. Introduction

Many online or local data sources provide powerful querying mechanisms but limited ranking capabilities. For instance, PubMed allows users to submit Boolean keyword queries on the biomedical publications database, but ranks the query results by publication date *only*. Similarly, the US Patent and Trademark Office (USPTO) allows Boolean keyword queries or searching patents but only ranks by patent date. Furthermore, job search databases, like the job search of LinkedIn, allow users to sort job listings by date or title (alphabetically), but not by IR relevance of the job posting to the submitted query. As a more recent example, the micro-blogging service Twitter offers a highly expressive Boolean search interface but ranks the results by date only. In most cases, these sources do not allow downloading and indexing of data or the size of the underlying database makes any comprehensive download [11], [12] an expensive operation.

Often, the user prefers a ranking other than the default (e.g., by date) provided by the source. For instance, a user of the PubMed or USPTO Web sites may sometimes prefer a ranking by relevance, measured by an Information Retrieval (IR) ranking function. Given that traditional IR ranking functions [14] like Okapi [15] and BM25 [13] implicitly assume disjunctive (OR) semantics, the naive approach would be to submit a disjunctive query with all query keywords, retrieve all the returned documents, and then rank them according to the relevance metric of choice. However, this would be very expensive due to the large number of results returned by disjunctive queries. For example, consider the query "immunodeficiency virus structure", an example query used to teach information specialists how to search the PubMed database [5]. Executing the corresponding disjunctive query "immunodeficiency OR virus OR structure" on PubMed returns 1,451,446 publication results. Downloading and ranking them is infeasible for an interactive query system, even if the source is on the local network. The problem becomes even more critical if we use the public web services provided by PubMed for programmatic (API) access over the web. Given the large overhead incurred when retrieving publications, PubMed imposes quotas on the amount of data an application can retrieve per minute, rendering infeasible any attempt to download large number of documents.

To overcome such problems, in this paper we present algorithms to compute the top results for an IR ranked query, over a source with a Boolean query interface but without any ranking capabilities (or with a ranking function that is generally uncorrelated to the user's ranking e.g., by date). A key idea behind our technique is to use a probabilistic modeling approach, and estimate the distribution of document scores that are expected to be returned by the database. Hence, we can estimate what are the minimum cutoff scores for including a document in the list of highly ranked documents. To achieve this result over a database that allows only query-based access of documents, we generate a querying strategy that submits a minimal sequence of conjunctive queries to the source. (Note that conjunctive queries are cheaper since they return significantly fewer results than disjunctive ones.) After every submitted conjunctive query we update the estimated probability distributions of the query keywords in the database

978-1-4244-5446-4/10/$26.00 © 2010 IEEE

and decide whether the algorithm should terminate given the user's results confidence requirement or whether further querying is necessary; in the latter case, our algorithm also decides which is the best query to submit next. For instance, for the above query "immunodeficiency virus structure", the algorithm may first execute "immunodeficiency AND virus AND structure", then "immunodeficiency AND structure" and then terminate, after estimating that the returned documents contain all the documents that would be highly ranked under an IR-style ranking mechanism. As we will see, our work fits into the "exploration vs. exploitation" paradigm [2], [9], [10], since we iteratively explore the source by submitting conjunctive queries to learn the probability distributions of the keywords, and at the same time we exploit the returned "document samples" to retrieve results for the user query. Our approach can also be extended and applied to other settings where the ranking is monotonic on a set of factors (query keywords in IR) and the source query interface is a Boolean expression of these factors.

II. Related Work

Top-k queries Theobald et al. [16] describe a framework for generating an approximate top-k answer, with some probabilistic guarantees. In our work, we use the same idea; the main difference is that we only have "random access" to the underlying database (i.e., through querying), and no "sorted access." Ilyas et al. [7] provide a survey of the research on top-k queries on relational databases.

Exploration vs. exploitation The idea of the exploitation/exploration tradeoff [2], [9], [10] (also called the "multi-armed bandit problem") is to determine a strategy of sequential execution of actions, each of which has a stochastic payoff. While executing an action we get back some (uncertain) payoff, and at the same time we get some information that allows us to decrease the uncertainty of the payoff of future actions.

Deep Web Our work bears some similarities to the problem of extracting data from the Deep Web [1] databases. For example, Ntoulas et al. [12] attempt to download the contents of a Deep Web database by issuing queries through a web form interface. [3], [8] characterize databases by extracting a small sample of documents that is then used to describe the contents of the database. In the experimental section, we compare against this "static sampling" alternative and demonstrate the superiority of the dynamic sampling technique, which dynamically generates estimates tailored to the query at hand.

III. Problem Definition

Query Model Consider a text database D with documents $d, \ldots, d_m$. The user submits a keyword query $Q = \{t_1 ... t_n\}$ containing the terms $t_1 ... t_n$. The answer to the query is a list of the top k documents; the documents are ranked according to a relevance score $score(Q, d)$, which estimates the relevance of a document d to the query Q.

The score of a document can be computed using any of the the well studied *tf.idf* scoring functions like BM25 and Okapi [13], [14], [15]. The key arguments of a *tf.idf* function are the term frequency (tf), the document frequency (df) and the document length (dl). The *term frequency* $tf(t, d)$ is the number of times that the word t appears in document d. The document frequency $df(t, D)$ is the number of documents in D that contain t. Hence, $score(Q, d) = F(tf, df, dl)$. At its basic form, the *tf.idf* ranking function is:

$$score(Q, d) = \sum_{t \in Q, d} tf(t, d) \cdot \ln \frac{|D| + 1}{df(t, D)} \qquad (1)$$

where $|D| = m$ is the size of the database D. In our experiments, we use the Okapi scoring function, although any other *tf.idf* function could be used. For simplicity though we use the basic *tf.idf* scoring function as the running example.

Data Source Model We assume that database D is only accessible through a Boolean query interface and we do not have direct access to the underlying documents. The query interface evaluates the Boolean query Q and returns the documents ranked using a non-desirable ranking function, e.g., by date (as is the case for PubMed and USPTO).

For instance, if the user query is Q=[*anemia, diabetes, sclerosis*], then we can submit to the data source queries q_1 = [*anemia AND diabetes AND sclerosis*], q_2 = [*anemia AND diabetes AND NOT sclerosis*], q_3 = [*diabetes OR sclerosis*], and so on. The returned results are guaranteed to match the Boolean conditions but the documents are not expected to be ranked in any useful manner.

Objective We want to devise a scheme for retrieving from D the top-k documents, ranked according to $F(tf, df, dl)$. The trivial solution is to send an extremely broad disjunctive query, returning all documents that have a non-zero $F(tf, df, dl)$ score. Then, we can retrieve the documents, examine their contents, and rerank them locally before presenting the results to the user. Unfortunately, this is a very time-consuming solution. Therefore, our objective is to construct a query sequence $q_1, q_2, \cdots, q_v$ of Boolean queries, that can be submitted to the database, retrieve as few documents as possible, and still contain all the documents that would be in the top-k results.

IV. Overview of Approach

As mentioned above, our approach is based on choosing the best sequence $q_1, q_2, \cdots, q_v$ of Boolean queries to submit to the data source, such that we retrieve the top-k ranked documents for Q. Of course, to select the best sequence of queries, we need to know some statistics about the type of documents retrieved by each query q_i. To get these statistics we need to sample the database through query-based sampling. So, through querying we are both retrieving documents to generate the necessary statistics and at the same time aim to retrieve documents that are in the top-k relevant documents. So, we can consider our approach as a case of "exploration vs. exploitation."

Even though we can use any Boolean query in our strategy, we only consider conjunctive Boolean queries as candidates,

given that a disjunctive query can be split to a set of conjunctive queries. Conjunctive queries provide a good query granularity and simplify the analysis below. Note that in practice we add negation conditions to the issued conjunctive queries in order to avoid retrieving the same results multiple times. For instance, if $Q = \{a, b\}$, after submitting $q_1 = a\ AND\ b$, we submit $q_2 = a\ AND\ NOT\ b$ instead of $q_2 = a$.

So, what are the goals of our querying strategy? Following Equation 1, we need to know the *tf* and *df* values for the terms in the database, to estimate the similarity score of a query to a document. Using these values, we can then estimate the overall *similarity score distribution* for all the documents in the database. Given the score distribution, we can compute how many documents in the database have score higher than the documents that we have seen so far.

The relatively easy part is the estimation of the *df* values. We can estimate these values in two ways: (a) We can send n queries to the database, one for each query term t_i, and compute the *df* value for each term. Note that the PubMed eUtils, which we use in our experiments, have a method to directly return the number of results (*df*) for a query. (b) We can use estimates of the *idf* (inverse *df*) values by using some other database with similar content (for example, using the Google Web 1T 5-gram collection[1]).

The more challenging part is the estimation of the *tf* values. We need to estimate the value of *tf* for each query term and for each document, that is, a total of $n \times |D|$ values. This is rather unrealistic without having direct access to the underlying database. So, we adopt a *query-based* probabilistic approach and we use the fact that term frequencies (*tf*) tend to follow a Poisson distribution within the documents of a database [16]. The more accurately we know the parameters of the distribution, the better we can estimate the document score distribution, and the better we can estimate how many documents should be in the top-k results but are still not retrieved.

Below, we describe briefly our strategies, giving the intuition behind each approach. We provide the complete theoretical analysis and the associated algorithms in the extended version of this paper [6].

One strategy for estimating the distribution parameter values is to generate a static document sample from the database and use this sample as a database summary for our estimations. However, we found that this *summary-based* strategy has low accuracy. The alternative, *query-based* strategy, relies on the exploitation-exploration framework, and combines sampling and query execution. In particular, our algorithms learn the *tf* distributions of the query keywords while the sequence of conjunctive queries are submitted. We account for the query-bias of the retrieved samples. That is, we estimate, given the retrieved documents for query t, how many empty documents we would have seen if we were performing random sampling. Now, assuming that we know the score distribution for Q of the documents, we can estimate the benefit that each issued query will generate: we can estimate the distribution of document scores (with respect to Q) for the documents retrieved by a conjunctive query q. Therefore, we can estimate the *benefit* of a query q, defined as the probability that a randomly selected document from the answer of q will have score higher than the k-th ranked score for Q among the documents retrieved so far.

To achieve that, we create a priority queue with all candidate queries q, ordered by expected benefit. We select the query at the top of the priority queue, retrieve documents, and based on the results we update the expected benefits of the other queries. Then, we pick the query with the next-highest expected benefit and so on. The algorithm terminates when the benefit (i.e., probability of retrieving a top-k document) drops below a user-specified probability constant P. That is, the algorithm terminates when every unseen result has probability less than P to be in the top-k answer. Note that P is provided by a domain expert to balance response time and accuracy, and hence users do not have to worry about it in practice. In the next sections we describe in detail our approach.

V. Experiments

We experimentally evaluate the performance and quality of the retrieval algorithms. We compare the Query-based probability estimation strategy to the Summary-based estimation strategy:

- *Baseline*: This algorithm submits the disjunction of all query keywords to the database, gets all results, computes their IR score, and finally return the top-k to the user.
- *Summary-based*: A sequence of conjunctive queries are submitted. The *tf* distributions are computed using a query-independent static database sample, as described in Section IV.
- *Query-based*: A sequence of conjunctive queries are submitted. The *tf* distributions are computed using the query-dependent, exploration and exploitation approach overviewed in Section IV.

Configuration: All experiments were run on a PC with a 2.5G Intel quad-core processor with 4G RAM running Windows XP SP2. The algorithms were implemented in Java.

Datasets: We ran our algorithm on the PubMed dataset, which can be remotely accessed through PubMed Web access utility services (*RemotePubMed*).[2] We only retrieve the abstracts of the articles since the body of many articles is missing from PubMed. Note that PubMed does not offer any form of relevance-based ranking. All results are ranked by date.

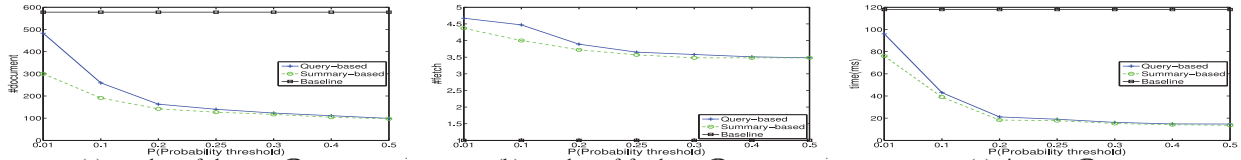
We used a subest of the English Test Questions from the TREC website[3] as our queries.

Quality Measure: We measure the quality of the algorithms as follows: we first execute the Baseline algorithm to compute the optimal top-k results. Then, we measure the quality of

[1] http://www.ldc.upenn.edu/Catalog/docs/LDC2006T13/

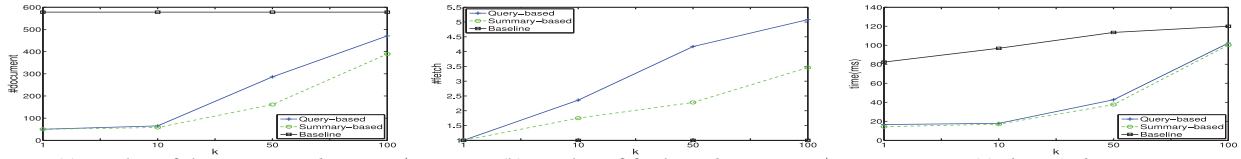
[2] http://www.ncbi.nlm.nih.gov/entrez/query/static/eutils_help.html

[3] http://trec.nist.gov/data/testq_eng.html



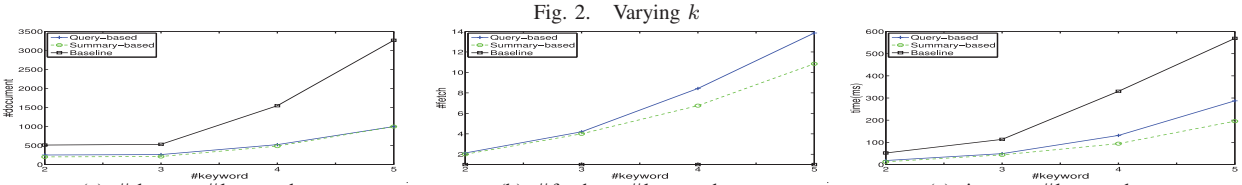
(a) number of doc. vs. $\mathcal{P}$ (b) number of fetch vs. $\mathcal{P}$ (c) time vs. $\mathcal{P}$

Fig. 1. Varying $\mathcal{P}$



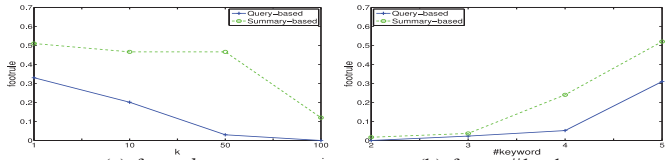
(a) number of documents vs. k (b) number of fetch vs. k (c) time vs. k

Fig. 2. Varying k



(a) #doc vs. #keywords (b) #fetch vs.#keywords (c) time vs. #keywords

Fig. 3. Varying #keywords



(a) fr vs. k (b) fr vs. #kwd

Fig. 4. Footrule vs. k and #kwd

Query-based and Block-based algorithms by comparing their top-k search results to this optimal list generated by the Baseline algorithm. We compare two top-k lists using the normalized top-k Spearman's Footrule metric [4].

In Figures 1, 2 and 3 we compare the three approaches for vaying user-specified results confidence $\mathcal{P}$, top-k and #keywords respectively.

Generally, we see that the Summary-based variant is slightly faster than the Query-based variant. On the other hand, Query-based is more accurate since its estimation strategy is better.

VI. Conclusions

We presented a framework and efficient algorithms to build a ranking wrapper on top of a documents data source that only serves Boolean keyword queries. This setting is common in various major databases today, including PubMed and USPTO. Our algorithm submits a minimal sequence of conjunctive queries instead of a very expensive disjunctive one. The query score distributions of the candidate conjunctive queries are learned as documents are retrieved from the source. Our comprehensive experimental evaluation on the PubMed database shows that we achieve order of magnitude improvement compared to the baseline approach.

Acknowledgments

Vagelis Hristidis was partly supported by NSF grant IIS-0811922 and DHS grant 2009-ST-062-000016. Panagiotis G. Ipeirotis was supported by the National Science Foundation under Grant No. IIS-0643846.

References

[1] M. K. Bergman. The Deep Web: Surfacing hidden value. *Journal of Electronic Publishing*, 7(1), Aug. 2001.

[2] D. A. Berry and B. Fristedt. *Bandit Problems: Sequential Allocation of Experiments*. Springer, 1985.

[3] J. P. Callan and M. Connell. Query-based sampling of text databases. *ACM Trans. on Information Systems*, 19(2):97–130, 2001.

[4] R. Fagin, R. Kumar, and D. Sivakumar. Comparing top k lists. In *SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 28–36, Philadelphia, PA, USA, 2003.

[5] R. C. Geer, D. J. Messersmith, K. Alpi, M. Bhagwat, A. Chattopadhyay, N. Gaedeke, J. Lyon, M. E. Minie, R. C. Morris, J. A. Ohles, D. L. Osterbur, and M. R. Tennant. NCBI advanced workshop for bioinformatics information specialists: Sample user questions and answers. Accessible at `http://www.ncbi.nlm.nih.gov/Class/NAWBIS/index.html`, 2002. Last revised on August 6th 2007.

[6] V. Hristidis, Y. Hu, and P.G. Ipeirotis. Efficient Ranked Queries on Sources with Boolean Query Interfaces. *NYU/CeDER Working Paper CeDER-09-05*. Available at `http://hdl.handle.net/2451/28302`, 2009.

[7] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4):1–58, 2008.

[8] P. G. Ipeirotis and L. Gravano. Distributed search over the hidden web: Hierarchical database sampling and selection. In *VLDB*, pages 394–405, 2002.

[9] J. Lee, J. Lee, and H. Lee. Exploration and exploitation in the presence of network externalities. *Management Science*, 49(4):553–570, Apr. 2003.

[10] W. G. Macready and D. H. Wolpert. Bandit problems and the exploration/exploitation tradeoff. *IEEE Transactions on Evolutionary Computation*, 2(1):2–22, Apr. 1998.

[11] J. Madhavan, D. Ko, L. Kot, V. Ganapathy, A. Rasmussen, and A. Y. Halevy. Google's deep web crawl. *PVLDB*, 1(2):1241–1252, 2008.

[12] A. Ntoulas, P. Zerfos, and J. Cho. Downloading textual hidden web content by keyword queries. In *JCDL*, 2005.

[13] S. E. Robertson, S. Walker, S. Jones, M. Hancock-Beaulieu, and M. Gatford. Okapi at trec-3. In *TREC*, 1994.

[14] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986.

[15] A. Singhal. Modern information retrieval: A brief overview. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 24(4):35–42, 2001.

[16] M. Theobald, G. Weikum, and R. Schenkel. Top-k query evaluation with probabilistic guarantees. In *VLDB*, pages 648–659, 2004.