ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Efficient Prediction of Difficult Keyword Queries over Databases

Gurramkonda Lakshmi Priyanka
P.G. Scholar (M. Tech),
Department of CSE,
Srinivasa Institute of Technology & Sciences,
Ukkayapalli, Kadapa, Andhra Pradesh.

K.Rajasekhar Reddy
Assistant Professor,
Department of CSE,
Srinivasa Institute of Technology & Sciences,
Ukkayapalli, Kadapa, Andhra Pradesh.

ABSTRACT:

Keyword queries on databases provide easy access to data, but often suffer from low ranking quality, i.e., low precision and/or recall, as shown in recent benchmarks. It would be useful to identify queries that are likely to have low ranking quality to improve the user satisfaction. For instance, the system may suggest to the user alternative queries for such hard queries. In this paper, we analyze the characteristics of hard queries and propose a novel framework to measure the degree of difficulty for a keyword query over a database, considering both the structure and the content of the database and the query results. We evaluate our query difficulty prediction model against two effectiveness benchmarks for popular keyword search ranking methods. Our empirical results show that our model predicts the hard queries with high accuracy. Further, we present a suite of optimizations to minimize the incurred time overhead.

Index Terms:

Query performance, query effectiveness, keyword query, robustness, databases

I.INTRODUCTION :

Keyword query interfaces (KQIs) for databases have attracted much attention in the last decade due to their flexibility and ease of use in searching and exploring the data. Since any entity in a data set that contains the query keywords is a potential answer, keyword queries typically have many possible answers. KQIs must identify the information needs behind keyword queries and rank the answers so that the desired answers appear at the top of the list. Unless otherwise noted, it refers to keyword query as query in the remainder of this project. Databases contain entities, and entities contain attributes that take attribute values. Some of the difficulties of answering a query are as follows: First, unlike queries in languages like SQL, users do not normally specify the desired schema element(s) for each query term.

For instance, query Q1: Godfather on the IMDB database (http://www.imdb.com) does not specify if the user is interested in movies whose title is Godfather or movies distributed by the Godfather Company. Thus, a KQI must find the desired attributes associated with each term in the query. Second, the schema of the output is not specified, i.e., users do not give enough information to single out exactly their desired entities. For example, Q1 may return movies or actors or producers. It is important for a KQI to recognize such queries and warn the user or employ alternative techniques like query reformulation or query suggestions. It may also use techniques such as query results diversification. To the best of our knowledge, there has not been any work on predicting or analyzing the difficulties of queries over databases. Researchers have proposed some methods to detect difficult queries over plain text document collections. However, these techniques are not applicable to our problem since they ignore the structure of the database. In particular, as mentioned earlier, a KQI must assign each query term to a schema element(s) in the database. It must also distinguish the desired result type(s).

II. RELATED WORK:

Prediction of query performance has long been of interest in information retrieval. It is invested under a different names query difficulty, query ambiguity and sometimes hard query. Keyword Searching and Browsing in Databases using BANKS [4] describe techniques for keyword searching and browsing on databases that we have developed as part of the BANKS system (BANKS is an acronym for Browsing ANd Keyword Searching). The BANKS system enables data and schema browsing together with keyword-based search for relational databases. BANKS enables a user to get information by typing a few keywords, following hyperlinks, and interacting with controls on the displayed results; absolutely no query language or programming is required. The greatest value of BANKS lies in near zero-effort web publishing of relational data which would otherwise remain invisible to the web.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

BANKS may be used to publish organizational data, bibliographic data, and electronic catalogs. Search facilities for such applications can be hand crafted: many web sites provide forms to carry out limited types of queries on their backend databases. For example, a university web site may provide form interface to search for faculty and students. Searching for departments would require yet another form, as would search for courses offered. Creating an interface for each such task is laborious, and is also confusing to users since they must first expend effort finding which form to use Efficient IR-Style Keyword Search over Relational Databases [2] A key contribution of this work is the incorporation of IR-style relevance ranking of tuple trees into our query processing framework.

In particular, our scheme fully exploits single-attribute relevance-ranking results if the RDBMS of choice has text-indexing capabilities (e.g., as is the case for Oracle 9.1, as discussed above). By leveraging state-of-the-art IR relevance-ranking functionality already present in modern RDBMSs, we are able to produce high quality results for free-form keyword queries. For example, a query [disk crash on a net vista] would still match the comments attribute of the first Complaints tuple above with a high relevance score, after word stemming (so that "crash" matches "crashed") and stop-word elimination (so that the absence of "a" is not weighed too highly).

III.STRUCTURED ROBUSTNESS ALGORITHM :

Algorithm shows the Structured Robustness Algorithm (SR Algorithm), which computes the exact SR score based on the top K result entities. Each ranking algorithm uses some statistics about query terms or attributes values over the whole content of DB. Some examples of such statistics are the number of occurrences of a query term in all attributes values of the DB or total number of attribute values in each attribute and entity set. These global statistics are stored in M (metadata) and I (inverted indexes) in the SR Algorithm pseudocode. SR Algorithm generates the noise in the DB on-the-fly during query processing. Since it corrupts only the top K entities, which are anyways returned by the ranking module, it does not perform any extra I/O access to the DB, except to lookup some statistics. Moreover, it uses the information which is already computed and stored in inverted indexes and does not require any extra index.

Algorithm1 CorruptTopResults(Q,L,M,I,N)
Input: Query Q, Top-K result list L of Q by ranking function g, Metadata M, Inverted indexes I, Number of corrupted iteration N.
Output: S R score for Q.
1: S R ← 0; C ← { }; // C caches λT, λS for keywords in Q
2: FOR i=1 → N DO
3: I′ ← I; M′ ← M; L′ ← L; // Corrupted copy of I, M and L
4: FOR each result R in L DO
5: FOR each attribute value A in R DO
6: A′ ← A; // Corrupted versions of A
7: FOR each keywords w in Q DO
8: Compute # of w in A′ by Equation
9: IF # of w varies in A′ and A THEN
10: Update A′, M′ and entry of w in I′;
11: Add A′ to R′;
12: Add R′ to L′;
13: Rank L′ using g, which returns L, based on I′, M′;
14: S R += Sim(L,L′); // Sim computes Spearman correlation
15: RETURN S R ← S R / N; // AVG score over N rounds

Algorithm

Algorithm shows the Structured Robustness Algorithm (SR Algorithm), which computes the exact SR score based on the top K result entities. Each ranking algorithm uses some statistics about query terms or attributes values over the whole content of DB. Some examples of such statistics are the number of occurrences of a query term in all attributes values of the DB or total number of attribute values in each attribute and entity set. These global statistics are stored in M (metadata) and I (inverted indexes) in the SR Algorithm pseudocode. SR Algorithm generates the noise in the DB on-the-fly during query processing. Since it corrupts only the top K entities, which are anyways returned by the ranking module, it does not perform any extra I/O access to the DB, except to lookup some statistics.Fig. 1.(a) shows the execution flow of SR Algorithm. Once we get the ranked list of top K entities for Q, the corruption module produces corrupted entities and updates the global statistics of DB. Then, SR Algorithm passes the corrupted results and updated global statistics to the ranking module to compute the corrupted ranking list. SR Algorithm spends a large portion of the robustness calculation time on the loop that re-ranks the corrupted results (Line 13 in SR Algorithm), by taking into account the updated global statistics. Since the value of K

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

(e.g., 10 or 20) is much smaller than the number of entities in the DB, the top K entities constitute a very small portion of the DB.the global statistics largely remain unchanged or change very little. Hence, we use the global statistics of the original version of the DB to re-rank the corrupted entities. If we refrain from updating the global statistics, we can combine the corruption and ranking module together. This way re-ranking is done on-the-fly during corruption. SGS-Approx algorithm is illustrated in Fig. 1.(b)

SYSTEM ARCHITECTURE:

Figure1.Execution flow of SR algorithm And SGS –Approx (a)SR algorithm (b)SGS –Approx.

IV.PREDICTION FRAMEWORK

4.1 Noise Generation in Databases

In order to compute SR, we need to define the noise generation model fXDB (M) for database DB. It will show that each attribute value is corrupted by a combination of three corruption levels: on the value itself, its attribute and its entity set. Now the details: Since the ranking methods for queries over structured data do not generally consider the terms in V that do not belong to query Q, we consider their frequencies to be the same across the original and noisy versions of DB. The corruption model must reflect the challenges about search on structured data, where we showed that it is important to capture the statistical properties of the query keywords in the attribute values, attributes and entity sets. We must introduce content noise (recall that we do not corrupt the attributes or entity sets but only the values of attribute values) to the attributes and entity sets, which will propagate down to the attribute values. For instance, if an attribute value of attribute title contains keyword Godfather, then Godfather may appear in any attribute value of attribute title in a corrupted database instance. Similarly, if Godfather appears in an attribute value of entity set movie, then Godfather may appear in any attribute value of entity set movie in a corrupted instance.

4.2 Ranking in Original & Corrupted Database With the mapping probabilities estimated as described above, the probabilistic retrieval model for semi-structured data (PRMS) can use them as weights for combining the score from each element into a document score, as follows:

$$P(Q|d) = \prod_{i=1}^{m} \sum_{j=1}^{n} P_M(E_j|q_i) P_{QL}(q_i|e_j)$$

Here, the mapping probability PM(Ej|w) is calculated and the element-level query-likelihood score PQL(w|ej) is estimated in the same way as in the HLM approach.

$$P_M(E_j|w) = \frac{P_M(w|E_j)P_M(E_j)}{P(w)} = \frac{P_M(w|E_j)P_M(E_j)}{\sum_{E_k \in E} P_M(w|E_k)P_M(E_k)}$$

$$P_{QL}(q_i|e_j) = (1-\lambda)P(q_i|e_j) + \lambda P(q_i|E_j)$$

The rationale behind this weighting is that the mapping probability is the result of the inference procedure to decide V. Basic Estimation Techniques:

Data sets:

The INEX data set is from the INEX 2010 Data Centric Track [14]. The INEX data set contains two entity sets: movie and person. Each entity in the movie entity set represents one movie with attributes like title, keywords, and year. The person entity set contains attributes like name, nickname, and biography. The SemSearch data set is a subset of the data set used in Semantic Search 2010 challenge [15]. The original data set contains 116 files with about one billion RDF triplets. Since the size of this data set is extremely large, it takes a very long time to index and run queries over this data set. Hence, we have used a subset of the original data set in our experiments. We first removed duplicate RDF triplets. Then, for each file in SemSearch data set, we calculated the total number of distinct query terms in SemSearch query workload in the file. We selected the 20, out of the 116, files that contain the largest number of query keywords for our experiments. We converted each distinct RDF subject in this data set to an entity whose identifier is the subject identifier. The RDF properties are mapped to attributes in our model. The values of RDF properties that end with substring —#type" indicates the type of a subject. Hence, we set the entity set of each entity to the concatenation of the values of RDF properties of its RDF subject that end with substring —#type". If the subject of an entity does not have any property that ends with substring —#type", we set its entity set to —UndefinedType".

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

We have added the values of other RDF properties for the subject as attributes of its entity. We stored the information about each entity in a separate XML file. We have removed the relevance judgment information for the subjects that do not reside in these 20 files. The sizes of the two data sets are quite close; however, SemSearch is more heterogeneous than INEX as it contains a larger number of attributes and entity sets.

Query Workloads:

Since we use a subset of the dataset from SemSearch, some queries in its query workload may not contain enough candidate answers. We picked the 55 queries from the 92 in the query workload that have at least 50 candidate answers in our dataset. Because the number of entries for each query in the relevance judgment file has also been reduced, we discarded another two queries (Q6 and Q92) without any relevant answers in our dataset, according to the relevance judgment file. Hence, our experiments is done using 53 queries (2, 4, 5, 11-12, 14-17, 19-29, 31, 33-34, 37-39, 41-42, 45, 47, 49, 52-54, 56- 58, 60, 65, 68, 71, 73-74, 76, 78, 80-83, 88-91) from the SemSearch query workload. 26 query topics are provided with relevance judgments in the INEX 2010 Data Centric Track. Some query topics contain characters —+" and —–" to indicate the conjunctive and exclusive conditions. In our experiments, we do not use these conditions and remove the keywords after character —–". Some searching systems use these operators to improve search quality.

Top-K results:

Generally, the basic information units instructured data sets, attribute values, are much shorter than text documents. Thus, a structured data set contains a larger number of information units than an unstructured data set of the same size. For instance, each XML document in the INEX data centric collection constitutes hundreds of elements with textual contents. Hence, computing Equation 3 for a large DB is so inefficient as to be impractical. Hence, similar to [13], we corrupt only the top-K entity results of the original data set. We re-rank these results and shift them up to be the top-K answers for the corrupted versions of DB. In addition to the time savings, our empirical results in Section 8.2 show that relatively small values for K predict the difficulty of queries better than large values. For instance, we found that K = 20 delivers the best performance prediction quality in our datasets.

Number of corruption iterations (N): Computing the expectation in Equation 3 for all possible values of _x is very inefficient. Hence, we estimate the expectation using N >0 samples over M(|A|× V). That is, we use N corrupted copies of the data. Obviously, smaller N is preferred for the sake of efficiency. However, if we choose very small values for N the corruption model becomes unstable.

VI.RESULTS:

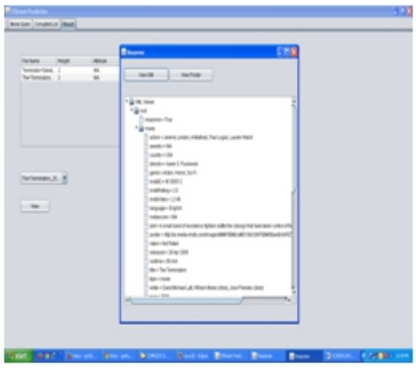

Figure 2

Figure 3

VII.CONCLUSION :

We introduced SR algorithm for difficult keyword queries over databases .The problem of predicting the effectiveness of difficult keyword queries over databases is introduced in this paper. We showed that the current prediction methods for queries over unstructured data sources cannot be effectively used to solve this problem. We set forth a principled framework and proposed novel algorithms to measure the degree of the difficulty of a query over a DB, using the ranking robustness principle. Based on our framework, we propose novel algorithms that efficiently predict the effectiveness of a keyword query.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

REFERENCES :

1. V..Hristidis, L. Gravano, and Y. Papakonstantinou, "Efficient IRstyle keyword search over relational databases," in Proc. 29th VLDB Conf., Berlin, Germany, 2003, pp. 850–861

2. Y. Luo, X. Lin, W. Wang, and X. Zhou, "SPARK: Top-k keyword query in relational databases," in Proc. 2007 ACM SIGMOD, Beijing, China, pp. 115–126.

3. V. Ganti, Y. He, and D. Xin, "Keyword++: A framework to improve keyword search over entity databases," in Proc. VLDB Endowment, Singapore, Sept. 2010, vol. 3, no. 1–2, pp. 711–722.

4. J. Kim, X. Xue, and B. Croft, "A probabilistic retrieval model for semistructured data," in Proc. ECIR, Tolouse, France, 2009, pp. 228

5. A. Nandi and H. V. Jagadish, "Assisted querying using instant-response interfaces," in Proc. SIGMOD 07 , Beijing, China,pp. 1156–1158.

6. O. Kurland, A. Shtok, D. Carmel, and S. Hummel, "A Unified framework for post-retrieval query-performance prediction," in Proc. 3rd Int. ICTIR, Bertinoro, Italy, 2011, pp. 15–26.

7. S. Cheng, A. Termehchy, and V. Hristidis, "Predicting the effec-tiveness of keyword queries on databases," in Proc. 21st ACMInt. CIKM , Maui, HI, 2012, pp. 1213-1222.

8. C. Hauff, L. Azzopardi, D. Hiemstra, and F. Jong, "Query perfor-mance prediction: Evaluation contrasted with effectiveness," in Proc. 32nd ECIR, Milton Keynes, U.K., 2010, pp. 204–216.