International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395 -0056
Volume: 02 Issue: 01 | Apr-2015 www.irjet.net p-ISSN: 2395-0072

EFFICIENT APPROACH FOR DETECTING HARD KEYWORD QUERIES WITH MULTI-LEVEL NOISE GENERATION

B.Mohankumar[1], Dr. P. Marikkannu[2], S. Jansi Rani[3], S. Suganya[4]

[1 3 4]*Asst Prof, Department of Information Technology, Sri Ramakrishna Engineering College, Coimbatore, India*
[2] *Head of the Dept, Department of Information Technology, Anna University, Regional Centre, Coimbatore, India*

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 the existing work, analyzes 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. However, in this system numbers of issues are there to address. They are, searching quality is lower than the other system and reliability rate of the system is lowest. In order to overcome these drawbacks, to perform the noise generation in three levels includes attribute level, attribute value level and entity set level in the database. This proposed system is well enhancing the reliability rate of the difficult query prediction system. In other words, this work is support these operators for efficient result. From the experimentation result, the proposed system is well effective than the existing system in terms of accuracy rate, quality of result***

Key Words: *Query Optimization, Query Performance and Keyword Query*

1. 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 *Q*1: *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, *Q*1 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).

1.1 Properties of Hard Keyword Query

The queries which are difficult to answer correctly are called hard keyword queries. The properties of hard keyword query are:

Less Specificity : If more entities match the terms in a query, the query is less specific and it is harder to answer properly.

Higher Attribute Level Ambiguity : Each attribute explains a different feature of an entity and defines the context of terms in attribute values of it. If a query matches different attributes in its candidate answers, it will have a more diverse set of potential answers in database, and hence it has higher attribute level ambiguity

Higher Entity set level Ambiguity : Each entity set contains the information about a different type of entities and defines another level of context (in addition to the context defined by attributes) for terms. Hence, if a query

matches entities from more entity sets, it will have higher entity set level ambiguity

2. RELATED WORKS

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. 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).
This scheme relies on the IR engines of RDBMSs to perform such relevance-ranking at the attribute level, and handles both AND and OR semantics. Unfortunately, existing query-processing strategies for keyword search over RDBMSs are inherently inefficient, since they attempt to capture all tuple trees with all query keywords. Thus these strategies do not exploit a crucial characteristic of IR-style keyword search, namely that only the top 10 or 20 most relevant matches for a keyword query –according to some definition of "relevance"– are generally of interest. The second contribution of this paper is the presentation of efficient query processing techniques for our IR-style queries over RDBMSs that heavily exploit this observation. As we will see, our techniques produce the top-k matches for a query –for moderate values of k– in a fraction of the time taken by state-of-the-art strategies to compute all query matches. Furthermore, our techniques are *pipelined*, in the sense that execution can efficiently resume to compute the "next-k" matches if the user so desires.

Predicting Query Performance via Classification [6] Here introduce new models and representations for estimating two important measures of query performance: query difficulty and expansion risk. This work brings together features from previous studies on query difficulty based on divergences between language models of the query, collection and initial results. Here extend this to include a model of expansion results from the expanded query. With these models and features, here compare the performance of two model representations: a low-dimensional pre computed topic representation and a much larger unigram language model over two standard Web collections. Here also develop a simple, effective method for deriving a topic representation, modeled as a distribution over ODP categories, of a query by estimating and combining pre-computed topic representations from the individual query terms. Here investigate using topic prediction data, as a summary of document content, to compute measures of search result quality. Unlike existing quality measures such as query clarity that require the entire content of the top-ranked results, class-based statistics can be computed efficiently online, because class information is compact enough to pre compute and store in the index. In an empirical study we compare the performance of class-based statistics to their language-model counterparts for two performance-related tasks: predicting query difficulty and expansion risk. Here findings suggest that using class predictions can offer comparable performance to full language models while reducing computation overhead.

3. PREDICTION FRAMEWORK

3.1 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.

3.2 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 $P_M(Ej|w)$ is calculated and the element-level query-likelihood score $P_{QL}(w|ej)$ 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 which element the user may have meant for a given query term.

3.3 Structured Robustness Algorithm

This compute the similarity of the answer lists using Spearman rank correlation. It ranges between 1 and −1, where 1, −1, and 0 indicate perfect positive correlation, perfect negative correlation, and almost no correlation, respectively. To computes the Structured Robustness score (*SR* score), for query *Q* over database *DB* given retrieval function

g: *SR(Q, g,DB,XDB)* = E{*Sim(L(Q, g,DB), L(Q, g,XDB))*}

where *Sim* denotes the Spearman rank correlation between the ranked answer lists.

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 λ_T, λ_S 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 10; // If $\lambda_{T,w}$ $\lambda_{S,w}$ needed but not in *C*, calculate and cache them
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 3.3: Structured Robustness Algorithm

Algorithm 3.3 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.

3.4 Approximation Algorithms

In this section, this paper proposes approximation algorithms to improve the efficiency of SR Algorithm. Our methods are independent of the underlying ranking algorithm.

Query-specific Attribute values Only Approximation (QAO-Approx): QAO-Approx corrupts only the attribute values that match at least one query term.

Observation 1: *The noise in the attribute values that contain query terms dominates the corruption effect.*

Observation 2: *The number of attribute values that contain at least one query term is much smaller than the numbers of all attribute values in each entity.*

Static Global Stats Approximation (SGS-Approx): *SGS Approx* uses the following observation:

Observation 3: *Given that only the top-K result entities are corrupted, the global DB statistics do not change much.*

4. ARCHITECTURE

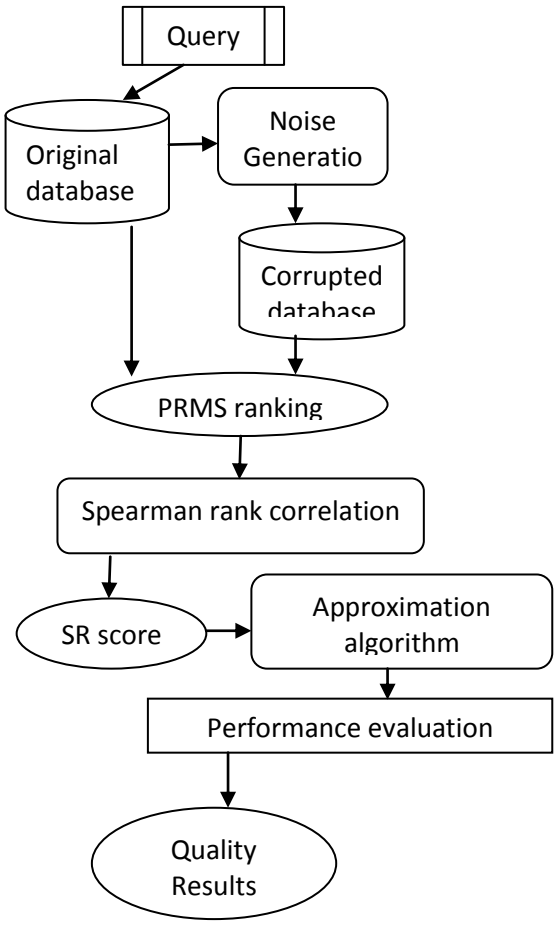

Fig -1: Work Flow Architecture

Form the fig 1, It shows the complete work flow of the proposed system and illustrate that how the system artichecture has been created for detecting the hard queries with multi-level noise generation and produes the quality result for the given query.

5. EXPERIMENTS AND RESULTS

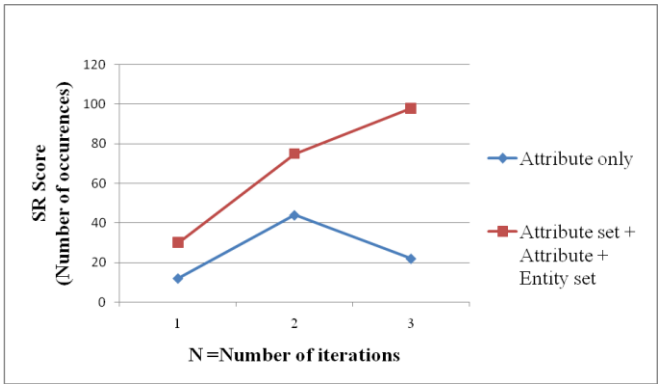

Fig 2 Comparison between Attribute onlyand Attribute set + Attribute + Entity setmethod

From the fig 2, It shows the comparison between the Attribute onlyand Attribute set + Attribute value + Entity set method where N is the number of iterations. In the Attribute only method, the SR score value is based on attribute set but in Attribute set + Attribute value + Entity set method, the SR score value is based on attribute value, attribute set and entity set. As number of iteration increases the SR score value decreases in Attribute only method but in Attribute set + Attribute value + Entity set method the SR score value linearly increases.

The comparison can be calculated as follows:

$$\text{Comparison} = \frac{\text{Average of number of occurrences in SR score}}{\text{Number of iterations}}$$

Table 1 Comparison between Attribute only and Attribute set + Attribute + Entity set method

Method	Iteration 1	Iteration 2	Iteration 3	Comparison in %
Attribute only	12	44	22	26 %
Attribute set + Attribute value + Entity set	30	75	98	64 %

From table 1, it shows that in Attribute only method the accuracy rate is 26% whereas in Attribute set + Attribute value + Entity set method the accuracy rate increased to 64%.

5. CONCLUSION

We introduced the novel problem of predicting the effectiveness of keyword queries over DBs. 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. Our extensive experiments show that the algorithms predict the difficulty of a query with relatively low errors and negligible time overheads.

REFERENCES

1. Shiwen Cheng, Arash Termehchy and Vagelis Hristidis (2014), 'Efficient Prediction of Difficult Keyword Queries over Databases', IEEE Transactions on Knowledge and Data Engineering, Vol. 26, no. 6, pp. 1507-1520.

2. Hristidis V., Gravano L. and Papakonstantinou Y. (2003), 'Efficient IR-style Keyword Search over Relational Databases', in Proc. 29th VLDB Conf., Berlin, Germany, pp. 850–861.

3. Ganti V., He Y. and Xin D. (2010), 'Keyword++: A Framework to Improve Keyword Search over Entity Databases', in Proc. VLDB Endowment, Singapore, Vol. 3, no. 1–2, pp. 711–722.

4. Bhalotia G., Hulgeri A., Nakhe C., Chakrabarti S. and Sudarshan S. (2002), 'Keyword Searching and Browsing in Databases using BANKS', in Proc. 18th ICDE, San Jose, CA, USA, pp. 431–440.

5. Zhou Y. and Croft B. (2006), 'Ranking Robustness: A Novel Framework to Predict Query Performance', in Proc. 15th ACM Int. CIKM, Geneva, Switzerland, pp. 567–574.

6. Collins-Thompson K. and Bennett P.N. (2010), 'Predicting Query Performance via Classification', in Proc. 32nd ECIR, Milton Keynes, U.K., pp. 140–152.

7. Shtok A., Kurland O. and Carmel D. (2009), 'Predicting Query Performance by Query-Qrift Estimation', in Proc. 2nd ICTIR, Heidelberg, Germany, pp. 305–312.

8. Zhao Y., Scholer F. and Tsegay Y. (2008), 'Effective Pre-retrieval Query Performance Prediction using Similarity and Variability Evidence', in Proc. 30th ECIR, Berlin, Germany, pp. 52–64.

9. Hauff C., Murdock V. and Baeza-Yates R. (2008), 'Improved Query Difficulty Prediction for the Web', in Proc. 17th CIKM, Napa Valley, CA, USA, pp. 439–448.

10. Hauff C., Azzopardi L., Hiemstra D. and Jong F. (2010), 'Query Performance Prediction: Evaluation Contrasted with Effectiveness', in Proc. 32nd ECIR, Milton Keynes, U.K., 2010, pp. 204–216.

