© 2016 IJSRSET | Volume 2 | Issue 2 | Print ISSN : 2395-1990 | Online ISSN : 2394-4099
Themed Section: Engineering and Technology

A Novel Method for Prediction of Difficult Keyword Queries

Sonal. G. Chopde, S. A. Murab
Department of Computer Science and Engineering, Jagdambha College of Engineering and Technology, Yavatmal, Maharashtra, India

ABSTRACT

Keyword queries are used to access data from databases. To improve the performance of querying system it would be useful to identify queries with low ranking quality. In this paper we analyze the characteristics of hard queries and propose a novel framework to measure the difficulty for a keyword query over a database. We evaluate our query difficulty prediction model against two effectiveness benchmarks for popular keyword search ranking methods. The ranking quality of the result provides a good user satisfaction. Our intensive experiments show that the algorithms predict the issue of a question with comparatively low errors and negligible time overhead.

Keywords: Query Performance, Query Effectiveness, Keyword Query, Robustness, Databases

I. INTRODUCTION

Keyword query interfaces for databases have attracted a lot of attention within the last decade because of their flexibility and easy use in looking and exploring the data. [1][5].Since any entity in a data set that contain the query keywords is a potential answer, keyword queries typically have many possible answer. Keyword queries should determine the information desires behind keyword queries and rank the answer so the required answers seem at the highest of the list [1] [6].Some of the difficulties of answering a query 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 are interested in movies whose title is Godfather or movies distributed by the Godfather company. Therefore, 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 [7]. For example, Q1 may return movies or actors or procedures. There are cooperative efforts to produce standard benchmarks and evaluation platforms for keyword search methods over databases. One effort is the data- centric track of INEX Workshop [8]. Wherever KQIs square measure evaluated over the well-known IMDB information set that contain structured information regarding movies and other people in show business. Queries were provided by participants of the workshop. Another effort is the series of Semantic Search Challenges at Semantic Search Workshop [9], where set is that the Billion Triple Challenge data set at http://vmlion25.deri.de.It's extracted from completely different structured data sources over the online like Wikipedia. The queries are taken from Yahoo! Keyword query log. User have to provided relevance judgment for both benchmarks. It is necessary for a KQI to acknowledge such queries and warn the user or use various techniques like question reformulation or question suggestion. It's going to additionally use techniques like question results diversification. For instance consider the query ancient Rome era over IMDB data set. Users would like to see information about movies that talks about the ancient Rome. For this query, the state-of-the-art XML search methods which we implemented return rankings of considerably lower quality than their average ranking quality over all queries. To the most effective of our data, there has not been any work on predicting or analyzing the difficulties of the queries over databases. Researchers have proposed some methods to detect difficult queries over plain text document collections [10][13]. However, these techniques aren't applicable to our drawback since they ignore the structure of the information. Above all, as mentioned earlier, a KQI should assign every question term to a schema element(s) within the information. It should additionally distinguish the specified result type(s). We tend to through empirical observation shows that direct diversifications of these techniques area unit ineffective for structured data.

We make the following contributions:

1. We introduce the problem of predicting the degree of the difficulty for queries over databases. We also analyze the reasons that make a query difficult to answer by KQIs.
2. We propose the Structured Robustness (SR) score, which measures the difficulty of a query based on the difference between the rankings of the same query over the original and noisy (corrupted) versions of the same database, where the noise spans on both the content and the structure of the result entities
3. We present an algorithm to compute the SR score, and parameters to tune its performance.
4. We introduce efficient approximate algorithms to estimate the SR score, given that such a measure is only useful when it can be computed with a small time overhead compared to the query execution time.
5. We show the results of extensive experiments using two standard data sets and query workloads: INEX and SemSearch. Our results show that the SR score effectively predicts the ranking quality of representative ranking algorithms, and outperforms non-trivial baselines, introduced in this paper.

II. METHODS AND MATERIAL

A. Literature Review

Researchers has been propose a methods to predict hard queries over unstructured text documents [10][13][17].We can broadly categorize these methods into two groups: pre-retrieval and post-retrieval methods. Pre-retrieval methods [14][18] predict the difficulty of a query that it does not utilize its result. These methods usually use the statistical properties of the term in the query to evaluate specificity, ambiguity, or term- relatedness of the query to predict its difficulty [19]. There are some examples of these statistical characteristics are average inverse document frequency of the query terms or number of documents that contain at least one query term [14]. We developed a method for predicting query performance by computing the relative entropy between a query language model and the corresponding collection language model. Post-retrieval methods utilize the results of a query to predict its difficulty and generally fall into one of the following categories.

Clarity-score-based

It is based on the concept of clarity score assume that users are concerned in a very few topics. Thus, sufficiently noticeable from other documents in the collection [10][14][15]. It is efficient than pre-retrieval based methods for text documents.[10] Some systems compute the distinguish ability of the queries results from the documents in the collection by comparing the probability distribution of terms in the results with the probability distribution of terms in the whole collection. If these probability distributions are relatively similar, the query results contain information about almost as many topics as the whole collection, thus, the query is considered becomes the difficult [10]. Too many successors has been proposed methods to improve the efficiency and effectiveness of clarity score [14], [15]. However, one requires domain knowledge about the data sets to extend idea of clarity score for queries over databases. Each topic in a database contains the entities that contain a similar subject. It is therefore hard to define a formula that partitions entities into topics as it requires finding an effective similarity function between entities. Such similarity function depends mainly on the domain knowledge and understanding users’ preferences [21]. For instance, distinct attributes may have distinct impacts on the degree of the similarity between entities

Ranking-score-based

Ranking score based methods defines the ranking score of the document returned by retrieval system for a user query. It estimates the similarity between a query and the document and defines the difficulty of a query by the difference between weighted entropy score of top ranked result and the score of other documents [16], [17]. Zhou and Croft shows that the information gained from a desired list of documents should be much more than the information gained from typical documents in the collection for an easy query. They measure the degree of the difficulty of a query by computing the difference between the weighted entropy of the top ranked results’ scores and the weighted entropy of other documents' scores in the collection [17]. Shtok *et al*. argue that whatever the amount of non-query-related information in the top ranked results must be negatively correlated with the deviation of their retrieval scores [16]. Using language modeling techniques, they show that the standard deviation of ranking scores of top-k results calculate the quality of the top ranked results effectively. We examine the query difficulty prediction accuracy of this set of methods on databases and show that our model outperforms these methods over databases.

Robustness Based:

Robustness-based methods defines how robust is the query over specific document. This method defines the degree of difficulty of query by considering the robustness of the ranking over two versions of data, Original version and corrupted version, and compares the top k results of same query over these two versions [12] Degree of difference between these results defines the degree of hardness of query. Some methods use machine learning techniques to study complex queries and its properties and then predict their hardness [22]. These methods are effective, if large amount and quality of data are available which are normally not available for many databases.

B. Properties of Hard Queries

As discussed above, it is well established that the more diverse the candidate answers of a query are, the more difficult the query is over a collection of the text documents. We extend this idea for queries over databases and propose three sources of difficulty for answering a query over a database as follows:

1) The more entities match the terms in a query, the less specificity of this query and it is harder to answer properly. For example, there are more than one person called *Ford* in the IMDB data set. If a user submits query *Q*2: *Ford*, a KQI must resolve the desired *Ford* that satisfy the user's information need. As opposed to *Q*2, *Q*3: *Spielberg* matches smaller number of people in IMDB, so it is easier for the KQI to return its relevant results.
2) Each attribute describes a different aspect 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. For instance, some candidate answers for query *Q*4: *Godfather* in IMDB some contain its term in their *title* and some contain its term in their *distributor*. For the sake of this example, we ignore other attributes in IMDB. A KQI must identify the desired matching attribute for *Godfather* to find its relevant answers. As opposed to *Q*4, query *Q*5: *taxi driver* does not match any instance of attribute *distributor*. Hence, a KQI already knows the desired matching attribute for *Q*5 and has an easier task to perform.
3) 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. For instance, IMDB contains the information about movies in an entity set called *movie* and the information about the people involved in making movies in another entity set called *person*. Consider query *Q*6: *divorce* over IMDB data set whose candidate answers come from both entity sets. However, movies about divorce and people who get divorced cannot both satisfy information need of query *Q*6. A KQI has a difficult task to do as it has to identify if the information need behind this query is to find people who got divorced or movies about divorce. In contrast to *Q*6, *Q*7: *romantic comedy divorce* matches only entities from *movie* entity set. It is less difficult for a KQI to answer *Q*7 than *Q*6 as *Q*7 has only one possible desired entity set. The aforementioned observations show that we may use the statistical properties of the query terms in the database to compute the diversity of its candidate answers and predict its difficulty, like the pre-retrieval predictors introduced in Section 2. One idea is to count the number of possible attributes, entities, and entity sets that contain the query terms to estimate the query specificity and ambiguity and use them to predict the difficulty of the query. The larger this value is the more difficult the query will be. We have shown empirically in Section 8.2 that such approach predicts the difficulty of queries quite poorly. This is because the distribution of query terms over attributes and entity sets may also impact the difficulty of the query. For instance, assume database *DB*1 contains two entity sets *book* and

movie and database *DB*2 contains entity sets *book* and *article*. Let term *database* appear in both entity sets in *DB*1 and *DB*2. Assume that there are far fewer movies that contain term *database* compared to books and articles. A KQI can leverage this property and rank books higher than movies when answering query *Q*8: *database* over *DB*1.

However, it will be much harder to decide the desired entity set in *DB*2 for *Q*8. Hence, a difficulty metric must take in to account the skewness of the distributions of the query term in the database as well. In Section 5 we discuss how these ideas are used to create a concrete noise generation framework that consider attribute values, attributes and entity sets.

III. RESULTS AND DISCUSSION

A. Implementation Details

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 [9]. 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". 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 [22], 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 [20]

B. Structure Robustness Algorithm

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.

Input: - Query Q, Top-K result list L of Q by ranking function g , Metadata M , Inverted indexes I , Number of corruption iteration N.

Output: - SR score for Q.

```
1. SR ← 0; C {}; //C catches λT, λS
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 keyword w in Q Do
8. Compute # of w in A' by Equation // If λT, w, λS, 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. SR+= Sim (L, L'); //Sim computers Spersman correlation
15. RETURN SR ← SR /N; //AVG score over N rounds
```

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 pseudo code.

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.

SR Algorithm loops every attribute value in each top-k result and test whether it must be corrupted. As noted one entity have hundreds of attribute values. We must note that the attribute values that do not contain any query term still must be corrupted. For a second and third level of corruption defined in equation. This is because their attribute or entity sets may contain some query keywords. This will largely increases the number of attribute value to be corrupted. For instance, for IMDB which has only two entity sets, SR Algorithm corrupts all attribute values in the top-k results for all query keywords. Second, ranking algorithms for DBs are relatively slow.SR Algorithm has to re-rank the top k entities N times which is time consuming.

C. Proposed Work

In this project, we analyze the characteristics of difficult queries over databases and proposed a novel method to detect such queries. This project takes advantage of the structure of the data to gain insight about the degree of the difficulty of a query given the database. This project introduces the problem of predicting the degree of the difficulty for queries over databases. This project also analyzes the reasons that make a query difficult to answer by KQIs. This project propose the Structured Robustness (SR) score, which measures the difficulty of a query based on the differences between the rankings of the same query over the original and noisy (corrupted) versions of the same database, where the noise spans on both the content and the structure of the result entities. This project presents an algorithm to compute the SR score, and parameters to tune its performance. Researchers have shown that this approach predicts the difficulty of a query more accurately than pre-retrieval based methods for text documents. Some systems measure the distinguish ability of the queries results from the documents in the collection by comparing the probability distribution of terms in the results with the probability distribution of terms in the whole collection. These methods usually use the statistical properties of the terms in the query to measure specificity, ambiguity, or term-relatedness of the query to predict its difficulty. Examples of these statistical characteristics are average inverse document frequency of the query terms or the number of documents that contains at least one query term.

IV. CONCLUSION

We introduced the novel problem of predicting the effectiveness of keyword queries over DBs. We showed that 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 Method to measure the degree of the difficulty of a query over a DB, using the ranking robustness principle. Our extensive experiments show that the algorithms predict the difficulty of a query with relatively low errors and negligible time overheads.

V. 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. 228–239.

[5] N. Sarkas, S. Paparizos, and P. Tsaparas, "Structured annotations of web queries," in Proc. 2010 ACM SIGMOD Int. Conf. Manage. Data, Indianapolis, IN, USA, pp. 771–782.

[6] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan, "Keyword searching and browsing in databases using BANKS," in Proc. 18th ICDE, San Jose, CA, USA, 2002, pp. 431–440.

[7] C. Manning, P. Raghavan, and H. Schütze, An Introduction to Information Retrieval. New York, NY: Cambridge University Press, 2008.

[8] A. Trotman and Q. Wang, "Overview of the INEX 2010 data centric track," in 9th Int. Workshop INEX 2010, Vugh, The Netherlands, pp. 1–32,

[9] T. Tran, P. Mika, H. Wang, and M.Grobelink,"Semsearch S10," in Proc.3rd Int. WWW Conf., Raleigh, NC, USA, 2010.

[10] S. C. Townsend, Y. Zhou, and B. Croft, "Predicting query performance," in Proc. SIGIR ’02, Tampere, Finland, pp. 299–306.

[11] A. Nandi and H. V. Jagadish, "Assisted querying using instantresponse interfaces," in Proc. SIGMOD 07, Beijing, China, pp. 1156–1158.

[12] E. Demidova, P. Fankhauser, X. Zhou, and W. Nejdl, "DivQ: Diversification for keyword search over structured databases," in Proc. SIGIR’ 10, Geneva, Switzerland, pp. 331–338.

[13] Y. Zhou and B. Croft, "Ranking robustness: A novel framework to predict query performance," in Proc. 15th ACM Int. CIKM, Geneva, Switzerland, 2006, pp.567-574.

[14] B. He and I. Ounis, "Query performance prediction," Inf. Syst., vol. 31, no. 7, pp. 585–594, Nov. 2006.

[15] K. Collins-Thompson and P. N. Bennett, "Predicting query performance via classification," in Proc. 32nd ECIR, Milton Keynes, U.K., 2010, pp. 140–152.

[16] A. Shtok, O. Kurland, and D. Carmel,"Predicting query performance by query-drift estimation," in Proc. 2nd ICTIR, Heidelberg, Germany, 2009, pp. 305–312.

[17] Y. Zhou and W. B. Croft, "Query performance prediction in web search environments," in Proc. 30th Annu. Int. ACM SIGIR, New York, NY, USA, 2007, pp. 543–550.

[18] Y. Zhao, F. Scholer, and Y. Tsegay, "Effective pre-retrieval query performance prediction using similarity and variability evidence," in Proc. 30th ECIR, Berlin, Germany, 2008, pp. 52–64.

[19] C. Hauff, L. Azzopardi, and D. Hiemstra, "The combination and evaluation of query performance prediction methods," in Proc.31st ECIR, Toulouse, France, 2009, pp. 301–312.

[20] V.Khalate, S.Gupta," An efficient forecasting of difficult keyword queries over databases,"vol.4, nov.2014.

[21] E. Yom-Tov, S. Fine, D. Carmel, and A. Darlow, "Learning to estimate query difficulty: Including applications to missing content detection and distributed information retrieval," in Proc.

[22] th Annu. Int. ACM SIGIR Conf. Research Development Information Retrieval, Salvador, Brazil, 2005, pp. 512–519.

[23] J. A. Aslam and V. Pavlu, "Query hardness estimation using Jensen-Shannon divergence among multiple scoring functions," in Proc. 29th ECIR, Rome, Italy, 2007, pp. 198–209.