# An Approach for Predicting Hard Keyword Queries

A Sangeetha
*Asst. Prof.*
*Department of Computer Science and Engineering,*
*CBIT, Hyderabad, Telangana State ,India*

P Sriharsha
*M.Tech. Student*
*Department of Computer Science and Engineering,*
*CBIT, Hyderabad, Telangana State ,India*

**Abstract: A query is a request for information retrieval from databases. Keyword query is referred as set of correlated words which collectively referred to a query. Keyword queries on databases provide easy access to data, but often suffer from low ranking quality. They are of low precision, and the system may suggest the user alternative queries for such hard queries. It would be useful to identify queries that are likely to have low ranking quality to improve the user satisfaction. A benchmark is a point of reference which is used for analyzing and evaluating the performance of a system or a database. Certain benchmarks are used for analyzing the performance of query , for improving the query from robustness. There have been collaborative efforts to provide standard benchmarks and evaluation platforms for keyword search methods over databases. One effort is the data-centric track of INEX Workshop Queries was provided by participants of the workshop. Another effort is the series of Semantic Search Challenges (SemSearch). The results indicate that even with structured data, finding the desired answers to keyword queries is still a hard task. A principled framework and novel algorithm implemented in this project, measure the degree of the difficulty of a query over a Database using the ranking robustness principle. Based on the framework, the algorithm produces the similarity scores which efficiently predict the effectiveness of a keyword query.**

**Keywords : Robustness , Database, Query, Hard Query.**

## I. INTRODUCTION

A database is a collection of information that is organized so that it can easily be accessed, managed, and updated. Database comprises of entities, entities posses attributes, attributes take attribute values. A query is a request for information retrieval from database. Keyword query is defined as set of correlated words which are collectively referred to a query. In order to retrieve the data , Keyword query interface (KQI's)[1] are useful for searching and exploring the data.

Users, unlike SQL queries, mainly for the information retrieval, do not usually cite the required idea schema elements for each query term. It is the responsibility of the KQI to find the required attributes associated to each term in the query. Moreover, users do not give enough information to select out from their desired entities.

As a result of the scenario, keyword queries on the databases are prone to low precision. Low precision occurs when results produced suffer from low ranking. KQI must recognize such queries and warn the user to employ alternatives like query reformulation or query suggestions. Such queries which have low precision are known as hard keyword queries or difficult keyword queries.

For predicting the difficult keyword queries, it is necessary to know some of the important properties of a query. The query may be under specified, where lower precision in a query. The second type is over precision, where specification of several keywords in a query is cited. The more different the answers of a query are, the more difficult the query is over a collection of data.

For example, query Q1: Godfather on the IMDB database (http://www.imdb.com) do not describe if the given 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 layout of the output is not specified, i.e., users do not give enough information to single out exactly their desired entities. The answers which are produced for the queries are vast in number and sometimes they are not ranked properly. Example consider a query as " ancient roman era" in IMDB database, the perspective behind the user who had posed the query would be to collect information of the movies which are based on roman era, but the ranking of the answers are low , that is they dont produce the exact movies list . Instead they may produce a production

unit name , or a dialogue from a movie etc , hence causing to low ranking, which are known as hard keyword queries.

From observations of benchmarks, two properties are analyzed for the hard queries. They are either the user queries which are posed are under specified, or they are over specified. Example, for under specified, "Carolina" as query. "Carolina" is under specified because it may be a city in USA, or it may be a name of a person, or even a name of restaurant somewhere else." Cricket player Yuvraj Singh best rating", consider this as over specification as the user gives all the information hints, which he want to get.

*1.1 Motivation*

While searching for the query, different types of keywords are combined to form a search. The results are produced where they may be relevant or they may be irrelevant. At sometimes the results are produced (example: In Google, the required results may be at the bottom of the irrelevant ones). To reduce this type of problem, we follow an algorithm which is known as structured robustness algorithm. By this we can at least reduce the irrelevant keywords and predict the required ones,

*1.2 Problem Statement*

A benchmark is a point of reference where performance of a system can be evaluated. Query performance and Query evaluation are some important features for analyzing the performance of information retrieval process. Standard benchmarks like INEX data centric workshop et al [6], and sem search.

Keyword queries are applied to the searching mechanisms of these benchmarks. They are useful in accessing the data, but they lack in the ranking of the precise answer. Often they are also recalled, for reconstructing query. Low precision occurs, as the system suggest the users for alternative queries instead of hard queries.

The characteristics of hard query are also not analyzed by these benchmarks, which can lead to understanding the hardness. The results indicate that even with structured data, finding the desired answers to keyword queries is still a hard task.

Keyword queries under the benchmark exhibit low rank suffering i.e low disclosing of the prescribed information in a result list. Performance is poor on the subset of query, i.e the part of the query results are not produced exactly

1.3 Solution

We measure degree of hardness of a keyword query over database by using the ranking robustness principle.

By using structured robustness algorithm proposed by Cheng et a[1] we calculate the rankings between the original and the corrupted versions of collected data , which is posed by a keyword query.

We can assure accurate results, when using the structured robustness algorithm. The results which are exact and, according to the user perspective are produced at the top results, thus reducing the hardness of a query.

The collection of information in the database can be in the form of structured and unstructured format. Structured database information is discrete in the form of rows and columns. Unstructured have less defined boundaries, small discrete objects. Certain methods have been proposed for predicting the hardness of a query. These methods have been employed on unstructured documents.

In the history methods are employed by the researchers over plain text document collections. However, these techniques are not applicable to the problem since they ignore the structure of the database. The methods are classified into two process pre-retrieval and post retrieval.

An early difficulty prediction technique, known as pre-retrieval [2] method is used for predicting the difficulty of a query without computing its results. 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. Average inversion term frequency is used as a measurement for statistical characteristics of a query term (OR) the frequency of number of documents that contain at least one query term. These methods assume that the more differentiable the query terms are, the easier the query is present.

Post-retrieval methods utilize the results of a query to predict its difficulty and generally fall into one of the following categories

1. Clarity scored based
2. Ranking score based
3. Robustness score base

*Clarity score based [3]* assume that users are interested in a very few topics, so they deem a query easy if its results belong to very few topics and therefore, sufficiently differentiable from other documents in the collection. 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. 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 difficult.

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 are about a similar subject. It is generally hard to define a formula that partitions entities into topics as it requires finding an effective similarity function.

*Ranking-score based* [4] of a document returned by the retrieval systems for an input query may estimate the similarity of the query and the document. Some recent methods measure the difficulty of a query. based on the score distribution of its results. Zhou and Croft et al[4] argue 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, they show that the standard deviation of ranking scores of top-k results estimates the quality of the top ranked results effectively.

*Robustness based* [5] are another group of post retrieval methods. Robustness based preretrieval is based on Ranking Robustness Principle, which argues that there is a (negative) correlation between the difficulty of a query and its ranking robustness in the presence of noise in the data. This robustness based principle as it is first applied in unstructured data it is called unstructured robustness principle. Mittendorf et al[5] has shown that if a text retrieval method effectively ranks the answers to a query in a collection of text documents, it will also perform well for that query over the version of the collection that contains some errors such as repeated terms . In other words, the degree of the difficulty of a query is positively correlated with the robustness of its ranking over the original and the corrupted versions of the collection. We call this observation the Ranking Robustness Principle.

The principle has been applied to predict the degree of the difficulty of a query over free text documents. It observes the similarities over rankings of query for the original and corrupted versions to predict the difficulty of a query over collection. They deem a query to be more difficult if its rankings over the original and the corrupted versions of the data are less similar. They have also shown that this approach is generally more effective than using methods based on the similarities of probability distributions from clarity scored based. This degree of prediction is especially important for ranking over databases.

Hence, we can use Ranking Robustness Principle as a domain independent proxy metric to measure the degree of the difficulties of queries.

A database mainly is in a structural format where the data or information is in form of rows and columns. To predict the hard queries, we analyze the property of these queries on database. It is well defined that the more diversified or more variety, the answers to a query are, the more hard or difficult the collection of query over text document.

### *1.1 Properties of hard queries*

The sources of difficulty for answering a query over database are analyzed

The more entities match the terms in a query, the less specificity of this query.

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 answers, it will have a more diverse set of potential answers in database, and hence it has higher attribute level ambiguity and it is harder to answer properly.

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.

## II. PROPOSED ALGORITHM

### *2.1 Algorithm for hard keyword queries*

Due to the drawback of difficulty of keywords, which produces low ranking, a structure has been proposed to understand the features of hard queries over the databases.

Some of the important properties of the difficulty in keywords had been surveyed it is well organized that the more diversified the answers of a query are, the more difficult the query is over a collection of the text documents. We expand this idea for queries over databases and propose three sources of difficulty for answering a query over a database as follows.

The more entities match the terms in a query, the less particularity of this query and it is harder to answer properly. For example: Tom is the name of an actor. when user submits the query: Tom, the KQI must

settle the desired tom according to the user request. Whereas the query: cuppola , matches a small number of people so it is easy to find the relevant answers.

Each attribute describes a different aspect of an entity and defines the context of terms in attribute values. If a query matches different attributes in its answers, it will have a more diverse set of potential answers in database, and hence it has higher attribute level ambiguity and it is harder to answer properly.

For example assume query: Godfather, present in IMDB, contains equivalent term in title and some contain its term in their distributor. A KQI finds the desired attribute for Godfather to get its précised answers. If we take query: taxi driver do not equal any occurrence of attribute distributor in IMDB. Hence, a KQI already knows the matching attribute for the query taxi driver and has an simple work to run.

Each entity set contains the knowledge about a different type of entities and describes another level of situation (in extension 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.

Example: In the IMDB database the data about the movie is present in the movie dataset and, actors in the actors database. If the query is "tragedy", then it is a difficult query because the KQI in IMDB must match the query term with the entities list of movies so as to search in the genre of tragedy movies . Also the query term must produce the results from the entities list of actors so that ,actors who are met with tragedy are also produced as results

In order to reduce the hardness Shiwen Cheng, Arash Termehchy, and Vagelis Hristidis et al [1] proposed a framework to understand the characteristics of hard queries over databases. We measure degree of hardness of a keyword query over database by using the ranking robustness principle.

By using structured robustness algorithm [1] we calculate the rankings between the original and the corrupted versions of collected data, which is posed by a keyword query.

We can assure accurate results, when using the structured robustness algorithm. The results which are exact with the high similarity scores are produced at the top of the list , according to the user perspective are produced at the top results thus reducing the hardness of a query.

The algorithm which is used for calculating the hardness of a query is structured robustness algorithm, the algorithm runs by the principle of ranking robustness principle [1] states that there is a opposed interrelationship between the hardness of a query and its ranking robustness in the presence of noise in the data. In other words, the degree of the difficulty of a query is positively correlated with the robustness of its ranking over the original and the corrupted versions of the collection. This principle is to predict the degree of the difficulty of a query over text documents. They compute the similarity between the rankings of the query over the original and the artificially corrupted versions of a collection.

*Step1: Corruption of database*

Corruption of structured data is the first step in the Ranking Robustness principle. For that, we create a database DB using a generative probabilistic model, with the help of the main contents which are terms, attribute values, attributes, and entity sets. A corrupted version of database is similar to the main database collected.

Let query be Q, retrieval function be g , the original database and corrupted versions are DB, DB'. The Database DB can be known as a triplet as it consists of **S** as entity set, **T** as attribute, **A** as attribute value, and **V** as the number of distinct terms in DB. Each of the attribute value **A** can be duplicated using V dimensional distribution. Xa=(Xa,1.....Xa,v), where Xa,j belongs to Xa, which is a random variable that shows the number of times of the term Wj in A. The probability mass function of Xa is

$$fXa(\underset{Xa}{\rightarrow})=Pr(Xa,1=xa,1.....Xa,v= x\ a,v)$$

The random variable XA=(X1,.....X|A|) models attribute value set A, where Xa belongs to XA is vector of size V that denotes the frequencies of terms in A. So XA is |A|×V matrix. Where |A| is the number of attribute values. The probability mass function of XA is

$$fXA(\underset{x}{\rightarrow})=fXA(\underset{X1}{\rightarrow}.....\underset{X|A|}{\longrightarrow}) = Pr(X1 = \underset{x1}{\rightarrow}.....X|A| = \underset{x|A|}{\longrightarrow})$$

Where $\underset{Xa}{\rightarrow} \in \underset{x}{\rightarrow}$ are vectors of size V that contain non negative integers. The domain of $x$ is all |A| × V matrices that contain non- negative integers, i.e M(|A| × V). Likewise X|T| and X|S| models attributes T and entity sets S, are corrupted in the same manner.

*Step2: Noise introduced in data*

Since XA, XT, XS are the corrupted contents of DB; we focus mainly on the noise introduced in the attribute values only. Noise is simply the inclusion of word in the attribute values of corrupted database $DB^i$ (|A| × V). To expand the idea of noise generation by an example, If in original database DB consists of attribute values A1 and A2 under attribute T, A1 have term W and A2 do not contain W. After the noise generation in the DB the term W exists under A1 and A2 attribute values.

Given query Q, let $\underset{x}{\rightarrow}$ be a vector that contains term frequencies for terms W ∈ Q ∩ V. The vector model is simplified by assuming that attribute values in DB and terms in Q ∩ V are independent.

$$f(XA)\overrightarrow{x} = \prod_{x\,a \in \overrightarrow{x}} f\,Xa(\overrightarrow{Xa}).$$
$$f(Xa)\overrightarrow{Xa} = \prod_{x\,a,j \in \overrightarrow{x}} f\,Xa,j(\,xa,j).$$

xj ∈ $\overrightarrow{xi}$ depicts the number of times wj appears in a noisy version of attribute value Ai and fXi,j (xj) computes the probability of term wj to appear in Ai xj times.

The calculation of structured robustness (SR score) is done using the similarity between the answer lists L and L' by spearman rank correlation. The rank ranges between 1 and -1. When we introduce the content noise to the attributes and entities it generates to the attribute values. For example attribute "NAME" contains keyword Titanic, the Titanic exists in the attribute value of attribute "NAME" in a corrupted database$DB^i$. If Titanic exists in attribute value of entity set "MOVIE", it must be appeared in the entity set "MOVIE" in corrupted database$DB^i$.

*Step3: Similarity score*

Structured robustness algorithm by cheng et al [1] calculates the SR score based on the top K result entities. The result lists are attribute values and we corrupt only the top-K entity results of the original data set. The re-ranking of the results will be done and they are moved to top answer lists as the top-K answers for the corrupted versions of DB.

Some of the statistical properties are used in the algorithms. These properties consist about the query terms or attributes values over the whole content of DB. 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 statistics are stored in metadata M, and inverted index I.

Algorithm: Structured robustness algorithm (corrupt top results)
Inputs: query Q, top K result list L of query Q, ranking function g, metadata M, Inverted Indexes I, Number of corruption iterations N.
Outputs: SR scores for a query Q.
(1) Let the SR score equal to 0, SR ← 0
(2) For 1 to all the number of iterations N ,
i = 1→ N DO
(3) Copying the corrupted lists ,and statistical properties M and I,
$I^i$ ←I; $M^i$ ←M; $L^i$ ←L.
(4) For each result R in ranking list L DO.
(5) For each attribute value A in R DO.
(6) $A^i$ ←A; copying the attribute values into corrupted database.
(7) For each keyword W of a query Q DO.
(8) Compute # of W in $A^i$. Hash set is used for storing the attribute values
and keywords of a query are both introduced into hash set for corrupted database DB'.
(9) IF # of W varies from $A^i$.and A then . If the values of attributes different from original and corrupted DB
(10) Update $A^i$.,M' and entry of the keyword W in $I^i$;
(11) ADD $A^i$.to $R^i$., Add $R^i$to $L^i$;
(12) Rank $L^i$ using ranking function g, which returns L, based on $I^i$, $M^i$;
(13) SR+ = sim(L, $L^i$); Structure robustness score is the similarity score of the result lists L and $L^i$ by spearman correlation.
(14) RETURN SR ← SR/N, average score over N rounds.
(15) END.

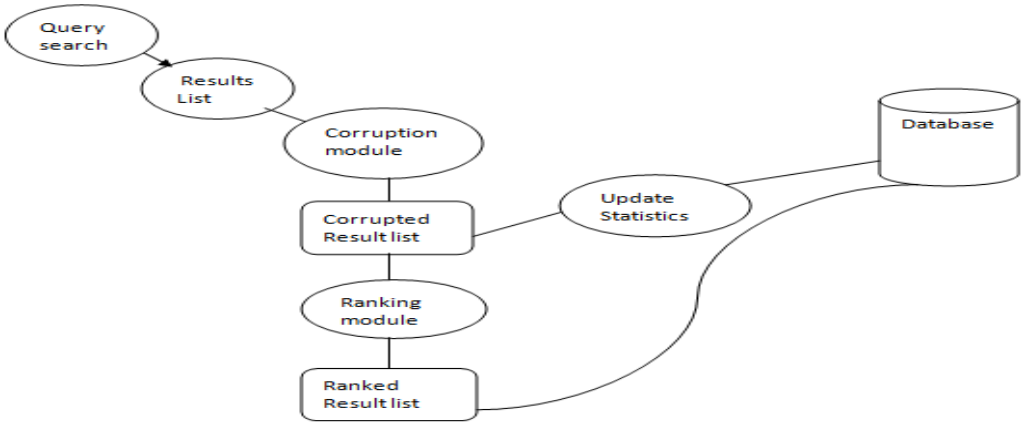


Fig1. Block diagram for hard keyword query

## III. EXPERIMENT AND RESULTS

The performance of the retrieval of data from the requested queries on the data set, by searching with noise and searching without noise. The time taken for system to return the information against particular queries are plotted and verified

### *3.1Searching without noise against runtime*

Table 1 Queries run time tabulated for search without noise

| INDEX | QUERY | RUNTIME |
|---|---|---|
| 1 | d009 | 47 |
| 2 | hein moni | 31 |
| 3 | d0009 | 49 |
| 4 | Sashi | 36 |
| 5 | d001 | 28 |
| 6 | d005 | 46 |
| 7 | 40000 | 422 |
| 8 | 53126 | 281 |
| 9 | Dimitri | 172 |
| 10 | senior eng | 188 |
| 11 | Mohan | 172 |
| 12 | Mansur | 176 |
| 13 | Msuda | 172 |
| 14 | 17-03-1985 | 187 |
| 15 | Staff | 172 |
| 16 | Maja | 313 |

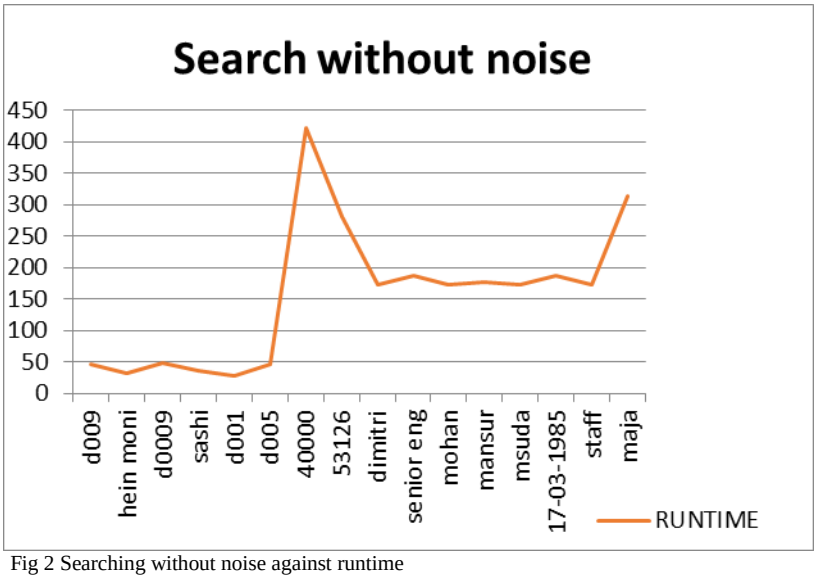


Fig 2 Searching without noise against runtime

Some queries are tested against the running time in form of Nano seconds. For the analysis of the performance from the searching without noise sixteen queries are tabulated with their runtime which is tested.

### *3.2 Searching with noise against runtime*

Table 2 Queries run time tabulated for search without noise

| INDEX | QUERY | RUNTIME |
|---|---|---|
| 1 | d009 | 28 |
| 2 | hein moni | 52 |
| 3 | d0009 | 16 |
| 4 | Sashi | 25 |
| 5 | d001 | 24 |
| 6 | d005 | 24 |
| 7 | 40000 | 176 |
| 8 | 53126 | 265 |
| 9 | dimitri | 156 |
| 10 | senior eng | 170 |
| 11 | mohan | 171 |
| 12 | mansur | 172 |
| 13 | msuda | 171 |
| 14 | 17-01-1985 | 172 |
| 15 | Staff | 172 |
| 16 | Maja | 219 |

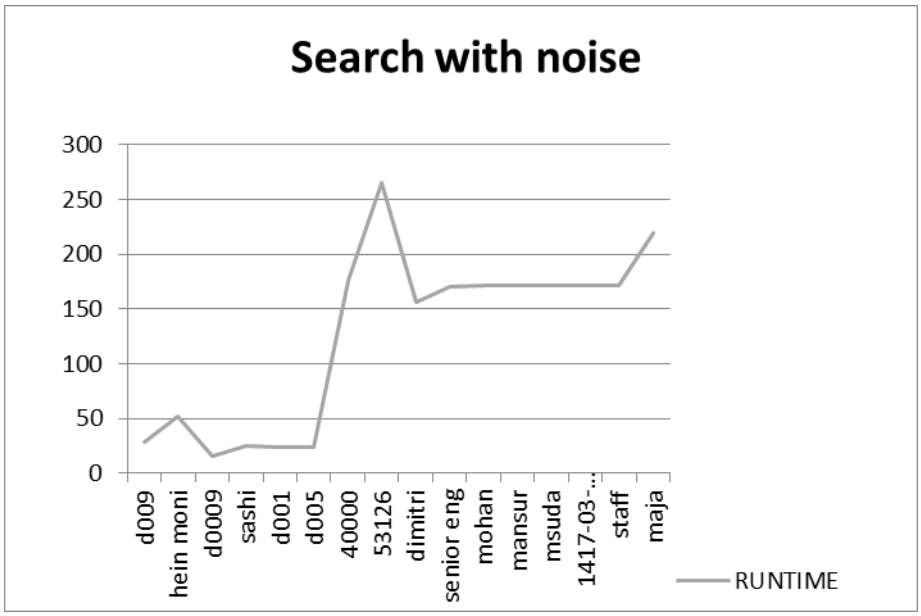


Fig 3 Searching with noise against runtime

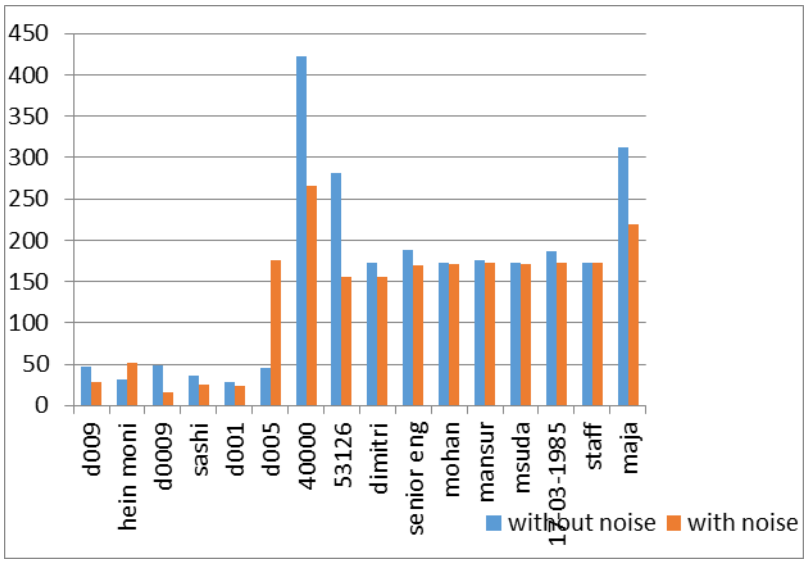


Fig 4 With and without noise search run time processing

## IV. CONCLUSION AND FUTURE WORK

Keyword queries over Databases are sometimes known for difficulty because of the less specific answers for the queries. In order to find the difficulty of keyword query we establish prediction of these queries. The main aim of this is to ensure effectiveness and also enhancement of the answers for the difficult queries.

The problem of level of hardness of a query therefore can be decreased by robustness principle. We implement a framework which consists of structure robustness algorithm to show the specific ranking by using similarity scores.

If a keyword is found in documents which are ranked according to the similarity scores, the same keyword can be found in lower documents according to the probability. So the same keyword found in lower documents can be produced resulting in ambiguity. So depending on the similarity scores ambiguity can be reduced by some metrics which can be developed in the future.

REFERENCES

[1] Shiwen Cheng, Arash Termehchy, and Vagelis Hristidis Efficient Prediction of Difficult Keyword Queries over Databases IEEE transactions on knowledge and data engineering volume 26, no 4, june 2014
[2] 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.
[3] Y. Zhou and W. B. Croft, "Query performance prediction in web search environments," in Proc. 30th Annual. Int. ACM SIGIR, New York, NY, USA, 2007.
[4] S. C. Townsend, Y. Zhou, and B. Croft, "Predicting query performance," in Proc. SIGIR'02, Tampere, Finland.
[5] Y. Zhou and B. Croft, "Ranking robustness: A novel framework to predict query performance," in Proc. 15th ACM Int. CIKM, Geneva, Switzerland, 2006.
[6] A. Trotman and Q. Wang, "Overview of the INEX 2010 data centric track," in 9th Int. Workshop INEX 2010,
[7] V. Ganti, Y. He, and D. Xin, "Keyword++: A framework to improve keyword search over entity databases," in Proceedings of VLDB Endowment, Singapore.
[8] V. Hristidis, L. Gravano, and Y. Papakonstantinou, "Efficient IRstyle keyword search over relational databases," in Proceedings of 29th VLDB Conf., Berlin, Germany, 2003.
[9] N. Sarkas, S. Paparizos, and P. Tsaparas, "Structured annotations of web queries," inProc. 2010 ACM SIGMOD Int. Conf. Manage. Data, Indianapolis, IN, USA.
[10] Arnab Nandi, H. V. Jagadish "Assisted Querying using Instant-Response Interfaces".