Effective Keyword Search over Relational Databases Considering keywords proximity and keywords N-grams

Sina Fakhraee
Dept. of Computer Science
Wayne State University
Detroit, MI 48202, USA
Fakhraee@wayne.edu

Farshad Fotouhi
Dept. of Computer Science
Wayne State University
Detroit, MI 48202, USA
Fotouhi@wayne.edu

***Abstract*— The current amount of text data in relational databases is massive and is growing fast. This increases the importance and need for non-technical users to be able to search for such information using a simple keyword search. Researchers have studied and addressed some of the issues with the efficiency and effectiveness of answering keyword queries in relational databases. In this paper we have summarized different factors affecting the effectiveness of the keyword search in relational databases which were studied in the early works. We have also identified two other important factors, namely keywords proximity and keywords N-grams that can further improve the search effectiveness when incorporated in the existing state-of-the-art information retrieval relevance ranking strategies for relational databases. Our experiments show that applying these two factors to the current ranking functions will improve the effectiveness of keyword search in relational databases.**

I. INTRODUCTION

The current amount of text data in relational databases is massive and is growing fast. This increases the importance and need for non-technical users to be able to search for information using a simple keyword search just as they would search for text documents on the web. Keyword search over relational databases (KSRDBs) enables ordinary users to query relational databases by simply submitting keywords without having to know any SQL or having any knowledge of the underlying structure of the data. Finding answers to keyword search over relational databases is a very challenging task, since good answers should be assembled by joining tuples from multiple relations across the database. Now, the effectiveness of keyword search over relational databases is even more of a challenge since unlike the text databases, relational databases have a much richer structure. In relational databases the search keywords are usually not just simply found in a single text attribute but they can be found in different text attributes of different relations, each of which having different degree of relevance to the search keyword(s).

The first couple of works in this area try to capture all the inter-connected tuples (i.e records from different relations joined on their primary-foreign keys) containing the exact Keywords and then rank the results purely based on the distance of the keyword-containing tuples from one another. This kind of approach is not very efficient, since most users are only interested in the first top-k search results. This approach is also not very effective because besides only the distances of the keyword-containing tuples from one another, other factors should be taken into account when ranking the answers. Modern relational databases have incorporated the state-of-the-art information retrieval (IR) relevance ranking techniques at the attribute level. Recent works in KSRDBs have exploited this functionality to answer keyword queries by identifying all database tuples that have a non-zero score for a given keyword search. Once these tuples are found, the first top-k tuples containing the keywords from each relation, if joinable, are joined via their primary-foreign key relationships and the ones which collectively contain the search keywords are presented to the users as the search results. Taking advantage of IR relevance ranking strategies employed by modern relational database management systems (RDBMSs) has improved both the efficiency and effectiveness of keyword search in relational databases. In this paper we have studied and summarized the IR style techniques used in recent research. Our key contribution is identifying two other important factors that can further improve the search effectiveness when incorporated into the IR relevance ranking strategies. These two factors are 1) the query keywords proximity, which is the overall distance of the keywords from one another in the value of the target text attribute and 2) the N-grams and in particular the quadgrams, of the query keywords in both the query itself and in the text attributes' values. Our experiments show that incorporating these two factors into the existing state-of-the-art ranking function will improve the effectiveness of KSRDBs. The remainder of this paper is organized as follows: Section II gives a brief overview of query result generation and describes basic concepts and definitions used in other literature for KSRDBs. Section III discusses related work and background of ranking used in KSRDBs. Section IV describes the keywords proximity and N-grams and how to incorporate them into the existing ranking function. Section V presents our experimental results. Section VI concludes our paper and gives direction for future work.

II. OVERVIEW OF QUERY RESULT GENERATION

In this section we describe how the results of a keyword query in relational databases are generated. Section *A* gives basic concepts and definitions used in the KSRDBs literature. Section *B* gives an algorithm to generate the results. The definitions and the algorithm given in sections *A* and *B* are adopted from previous works [2,4].

A. BASIC CONCEPTS AND FRAMEWORK

Below there are a few terms and notations defined in the KSRDBs literature [2,4] that we will use in next section and for that purpose we restate them below.

Definition 1 A *tuple tree T* also referred to as inter-connected tuples is a tree of tuples where for a tuple $t_i \in T$ adjacents to a tuple $t_j \in T$ where t_i is a tuple in R_i and t_j is a tuple in R_j , there is an edge from R_i to R_j in G, the schema graph of the relational database.

Definition 2 A *keyword query Q* is a set of keywords K_1, K_2,…, K_n entered by the user.

Definition 3 A *Query result R* is a set of *tuple trees* where for each *tuple tree T* each keyword k_i must appear in at least one tuple t_i of *T*.

Definition 4 A *Master index* is an inverted list that relates each keyword that appears in the database with a list of locations in the database which are recorded as row-column pairs.

Definition 5 A *tuple set* of relation R_i with respect to a keyword k_j denoted by $R_i^{k_j}$ is a set of tuples in R_i that contain k_j in at least one of their text attributes.

Definition 6 A *free tuple* set of relation R_i denoted by R_i^F is a set of all the tuples of relation R_i.

Definition 7 A *joining network of tuple sets J* is a tree of tuple sets where for a tuple set $R_i^{k_1} \in J$ adjacents to a tuple set $R_j^{k_2} \in J$, there is a corresponding edge (R_i, R_j) in G.

Definition 8 A *Candidate network* also referred to as *join tree* is a joining network of tuple sets that is used to generate answers to a keyword query.

B. QUERY RESULT GENERATION ALGORITHM

To generate answers for a keyword query *Q* given by the user three modules are involved. The first module in the pipeline is the master index which inputs the keyword query *Q* and returns a set of tuple sets for each relation with respect to each keyword. The second module in the pipeline is the candidate network generator which inputs the keyword query *Q*, set of tuple sets, free tuple sets and the schema graph of the relational database and outputs the set of candidate networks. The last module in the pipeline is the execution engine which inputs the set of candidate networks and outputs the set of answers to the keyword query by executing the actual SQL statements corresponding to each candidate network.

III. RELATED WORK AND BACKGROUND IN RANKING

Ranking of the query results has been addressed by previous works in keyword search over relational databases. The approaches taken by the early works [2,4] in this area were mostly based on the size of the answer tuple trees which is, given a query *Q* the score assigned to an answer tuple tree *T* is:

$$score(T,Q) = \frac{1}{size(T)} \tag{1}$$

where *size(T)*is the number of tuples in tuple tree *T*. In recent years the IR community has developed state-of-the-art ranking techniques that can be leveraged when ranking results for keyword search in relational databases. Luckily, modern relational databases have already incorporated the state-of-the-art IR relevance ranking functionality for individual text attributes. [1] exploited this feature to define their ranking function which has two sub-functions, namely *Score* and *Combine* as defined below:

$$Score(d,Q) = \sum_{k \in Q \cap d} \frac{1+\ln(1+\ln(tf))}{(1-s)+s\frac{dl}{avdl}} . ln\frac{N}{df} \tag{2}$$

where, *Score(d,Q)* is the relevance score with respect to the keyword query *Q* determined by an IR engine for a single text attribute *d* which is viewed as a text document. *k* is a single keyword in *Q*, *tf* is the term frequency of *k* in the value of text attribute *d*, *df* is the document frequency for keyword *k* which is the number of tuples in *d*'s relation with *k* appearing in *d*'s value, *dl* is the document length which is the number of characters of the value of *d*, *avdl* is the average length of the value of text attribute *d*, *N* is the number of tuples in *d's* relation and *s* is a constant number of value usually 0.2. Now, let *D* be the set of all the text attributes of an answer tuple tree *T*. The score assigned to *T* with respect to the query *Q* is calculated using the aggregate function *combine* as follows:

$$\begin{aligned} Score(T,Q) &= Combine\big(Score(D,Q), size(T)\big) \\ &= \frac{\sum_{d_i \in D} Score(d_i,Q)}{size(T)} \end{aligned} \tag{3}$$

[3] identified four important normalization factors that can improve the keyword search effectiveness:

1) In Formula 3, using the raw *size(T)* can be sub-optimal and might not rank the relevant answers accurately. [3] proposed a normalized *size(T)* denoted by *Nsize(T)* as defined below:

$$Nsize(T) = (1-s) + s * \frac{size(T)}{avgsize} \tag{4}$$

where *avgsize* is the average *T* size across all the answer tuple trees.

2) The second issue addressed by [3] is the average document length *avdl* in Formula 2. As discussed in that paper, the current definition of *avdl* only considers average text attribute length within the local text attributes. However, each text attribute has its own *avdl* which could be very different from another text attribute, consequently this could affect the relevance ranking of the answer tuple trees from different candidate networks. To resolve this issue, [3] proposed a normalized *avdl* denoted by *Navdl* which considers average text attribute's length within both local and global document collection as defined below:

$$Navdl = \frac{avdl}{(1+\ln(avdl))} \tag{5}$$

3) The third issue addressed by [3] is with the document frequency *df* and total number of tuples *N* in Formula 2. As discussed in [3], the same keyword term might have very different document frequencies in different document collections. Also *N* for different relations could be very different. To resolve these issues, [3] proposed global document frequency df^g and global document count N^g. df^g is the total number of text attributes' values containing the keyword and N^g is the total number of all text attributes' values in the entire database.

Due to the lack of space we refer the user to [3] for the fourth normalization factor. The final scoring function for tuple tree *T* with respect to query *Q* is obtained by replacing *size(T)* with *Nsize(T)*, *avdl* with *Navdl*, *df* with df^g and *N* with N^g. Comparing to the early works of [1,2], this ranking function has significantly improved the effectiveness of keyword search over relational databases.

IV. TWO IMPORTANT FACTORS TO FURTHER IMPROVE THE RANKING FUNCTION

The original IR-style relevance ranking for an individual text-attribute and all the proposed normalization factors, were primarily based on the different keywords' statistics such as local and global text-attribute frequencies, inverse local and global text-attribute frequencies and text-attribute length. We have identified two other important factors, namely keywords proximity and keywords n-grams that once incorporated into the ranking function, can improve the search's effectiveness.

A. Query Keywords Proximity

An important factor that should be incorporated into the existing relevance ranking function for relational databases is the keyword proximity which rewards a text attribute where the matched keywords are in the smallest proximity from one another. Previous works in KSRDBs [7, 8] have taken into account and introduced the notion of keyword proximity search mostly for inter-collection proximity (i.e. the distance between keywords found in different text attributes across the relational database). This has been done by assigning different weights to the nodes containing the keywords and the edges connecting them and searching for the minimum connected tree which collectively contain all the keywords. These approaches have significantly improved the search effectiveness. In this paper we consider IR-style keyword proximity search within a document and have adopted it for intra-collection keywords proximity search in relational databases (i.e. the distance between keywords found within a target text attribute). This is crucial when querying databases with long-string text attributes such as movie summaries, memos, product descriptions etc. We have used the definition for keyword proximity proposed in [5] as follows:

Definition 9 [5] defines *minimum pair distance proximity* as the smallest distance of all the pairs of distinct matched query keywords. It is denoted by *MinDist*(*Q*,*D*) and read as minimum keyword pairs distance of document *D* with respect to query *Q*. The formal definition is as follows:

$$MinDist(Q,D) = min_{k1,k2 \in Q \cap D}\{Dist(k1,k2)\} \quad (6)$$

where $k1 \neq k2$ and $Dist(k1,k2)$is the length of the shortest segment between *k1* and *k2* (i.e. *k1*-*k2* segment).

i. Incorporation of Keyword Proximity into Relevance Ranking Function

As mentioned in the previous section, the small proximity of matched query keywords in a document should reward that document by promoting its relevance ranking score. We cannot simply add proximity measure values to the values of the ranking function since these two quantities are not comparable as explained in [5]. Therefore, the proximity function, *MinDist*, should be transformed to a function that produces values that are comparable with relevance ranking scores and would reasonably impact the relevance values. Let γ be the transformation function and $\tau(Q,D)$ be the new transformed function also referred to as adjustment factor by [5]. Therefore, we will have: $\tau(Q,D) = \gamma(MinDist(Q,D))$. As proposed by [5] there are two criteria that must be met by the new transformed function$\tau(Q,D)$:

1) As mentioned above $\tau(Q,D)$ should positively impact the relevancy of the document to the query by promoting its relevance ranking score (i.e. the smaller the *MinDist* the larger $\tau(Q,D)$).

2) The effect of *MinDist* on τ should drop quickly as the distance gets smaller past some point and its effect should become constant as the distance becomes larger beyond some point. These two constraints lead to the following definition for τ defined by [5]:

$$\tau(Q,D) = \log(\alpha + \exp(-MinDist(Q,D))) \quad (7)$$

We have adopted this definition for τ as our adjustment factor to be added directly to the ranking function, Formula 2. As we will see in the experiment section the new ranking function has improved the effectiveness of the keyword search over relational database.

B. Keywords N-grams

Many users searching for information using keyword search might misspell the query term or might only know the partial spelling of the keyword(s). More importantly these keywords might have been misspelled in the target text attributes' values that the search is performed against. To demonstrate these two problems, consider the following three examples:

1) Assume a user is searching for action movies featuring actor *John Travolta* and is not sure about the correct spelling of the name of the actor. Therefore, he might perform his search by typing for example keyword sequences such as "*Actions Travelta*" or "*Actions Traveltha*" instead of "*Actions Travolta*". This could result in the failure of finding the intended inter-connected tuples depending on which SQL predicate (such as *LIKE*, *CONTAINS* or *FREETEXT*) was used to implement the search system.

2) Assume a user is searching for thriller movies featuring a German actress *Martina Gedeck*. If the targeted movie database has inconsistencies in how the actress name has been spelled (e.g. *Gedack*, *Gedek*, etc.), this could negatively impact the effectiveness of the keyword search as the *tf* and *idf* factors of Formula 2 will not be accurate due to possible mismatching between query keywords and the keywords in the text attributes' values.

3) Similar to the second problem above, if the values of a target text attribute contains different variations of the same verb or noun (e.g. verb category and its variations such as categories, categorization, etc.), this could also negatively impact the effectiveness of the keyword search as the *tf* and *idf* factors of Formula 2 will not be accurate due to possible mismatching between query keywords and the keywords in the text attributes' values.

To address these issues we have computed the N-grams of the keywords both in the values of the target text attributes and in the query itself and incorporated them to the ranking function as follow:

We incorporated the N-grams (in particular the quadgrams) of the keywords to the Formula 2 by updating the *tf* and *idf* not only when we encounter the exact search terms, but also when we encounter the variations of the corresponding terms generated by the quadgrams. Our experiments show that the modified version of Formula 2 will further enhance the keyword search effectiveness. The final ranking algorithm is shown in Figure 1. To decide which N-gram to use, one should consider the nature of the text, language in which the text is written and other factors. Even though shorter N-grams such as unigrams, bigrams and trigrams could perform very well on keyword matching, based on various experiments performed using our dataset and sample queries, we decided to only consider quadgrams because almost always a match between the first quadgram of a query keyword and the first quadgram of a term in the text attribute was a true match (i.e. the two terms were related or varied slightly due to misspellings).

Algorithm2: Relevance Ranking Algorithm
Input: Keyword query *Q*, a set of answer tuple trees *T*'s
Output: A ranked set of answer tuple tree.
1: *T-Set //* set of answer tuple trees *T*'s
2: *T-Ranked-Set //* a priority queue to store the set of
3: // ranked answer tuple trees *T*'s
4: *Q* // query keywords
5: *k_Quadgrams* //a set of all the quadgrams for keyword *k*
6: *For each* k_i in *Q*
7: computeQuadgrams(k_i)
8: k_i*_Quadgrams.update*
9: *end for*
10: *For each* k_i in each text_attribute
11: computeQuadgrams(k_i)
12: k_i*_Quadgrams.update*
13: *end for*
14: *For each T* in *T-Set*
15: *T-Ranked-Set* .push(*T, Score'*(*T*,*Q*)) // *Score'*(*T*,*Q*) is
16: // the modified version of Formula 3 after applying
17: // Quadgrams and $\tau(Q, d)$ to Formula 2.
18: *end for*
19: *Return T-Ranked-Set*

Figure 1. Relavance Ranking Algorithm.

V. EXPERIMENTS

We used the IMDB [9] dataset to perform our experiments. We designed and implemented a relational database corresponding to the IMDB schema and populated the database with a small portion of raw text files downloaded from IMDB. In our schema we included *plot_summary* text attribute which has long string values in order to be able to create queries for evaluating the effect of *Keywords Quadrams* and *keywords proximity* which are both more effective on text attributes containing long strings. The IMDB schema we used is shown in *Table I*.

TABLE I. IMDB DATASET

IMDB Schema	# of Rows
Actors(actorID, actor)	2000
Directors(directorID, director)	1200
Movies(movieID, title, directorID, summaryID)	4000
Cast(movieID, actorID)	8345
MovieCategories(movieID, genre)	6126
PlotSummary(movieID, summary)	4000
ActorPlay(*actorID, character, movieID*)	7310

When populating our database we purposely misspelled the names of some of the actors/directors/characters/titles and in the plot summary of the movies, we changed some of the terms to different forms (but all generated from the original terms). We created two types of queries; 1) *Type I* queries, targeting both short and long text attributes. 2) *Type II* queries, only targeting short text attributes (Please see *Table II* below). We then formulated 50 queries, 25 per each type. To assess the effectiveness of our approach with comparison with the previous works we used two measures: 1) Number of top-1 search results that are relevant denoted by #Rel in *Table III*. It shows how well the system retrieves one relevant answer. This metric is used for ranking tasks in which the user is looking for a single or a very small set of relevant answers in a large collection [6]. We chose this metric to evaluate *type I* queries since the user's primarily intention is to find a single movie. For example a user searching for a particular movie which has forgotten the name for, but remembers the genre of the movie and knows what the movie is about, would perform the search by entering the genre of the movie and few keywords describing the movie (e.g. query 4 in table2). 2) 11-point precision/recall (i.e. precision at recall level of 0.1). This measure shows the effectiveness of our system in retrieving *top10* answers. We chose this metric to evaluate *type II* queries since the user's primarily intention is to find a set of relevant movies. For example a user searching for a set of movies in a certain category in which a particular actor has played, would perform the search by entering the genre of the movie and the name of the actor (e.g. query 7 in table2).

TABLE II. QUERY TYPES

	Type I queries		Type II queries
1	Summary	7	actor, genre
2	Summary, director	8	actor, director
3	Summary, actor	9	director, genre
4	Summary, genre	10	actor, director, genre
5	Summary, actor, genre	11	actor, character
6	Summary, character		

In order to identify the relevant answers in our database we use pooled relevance judgment used in [3] as follows; We ran all four algorithms for each query (*BA*: base algorithm Formula 3, *BA+KP*: base algorithm + keywords proximity, *BA+KQ* = base algorithm + keywords quadgrans and *BA+KP+KQ* = base algorithm + keywords proximity + keywords quadgrans) and merge their *top20* results. We then manually judged and selected relevant results for each query out of the 80 candidate results. We chose pooled relevance judgment as our standard for evaluation because only the users can determine if a search result satisfies the query's need or not.

To evaluate the effectiveness and impact of keyword proximity and keyword quadgrams, we purposely submitted 12 out of 25 queries containing misspellings for *type II* queries. For *type I* queries, we submitted 12 out of 25 queries containing misspellings and phrases with different keywords' variations. *Table III* shows the number of top-1 search results for *type I* queries for each algorithm and Figure 2 shows the 11-point precision/recall graph for *type II* queries for *BA* and *BA+KP+KQ* algorithms.

TABLE III. IMPACT OF KEYWORDS PROXIMITY AND QUADGRAMS ON NUMBER OF TOP-1 RESULT

	BA	*BA+KP*	*BA+KQ*	*BA+KP+KQ*
#Rel	8	12	13	16

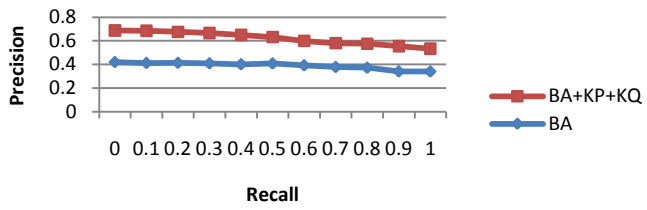

Figure 2. 11-point Precision/Recall.

We can see from *Table III* that *BA+KP+KQ* algorithm outperformed the base algorithm *BA* for *type I* queries. We observed that many of the relevant answers to the queries targeting the *plot summary*, which contained misspelled keywords or keywords which were located apart from one another, were not in the top-1 search results for *BA*, and in fact they were not even in the top-10 results. We can also see from Figure 2 that *BA+KP+KQ* algorithm outperformed the base algorithm *BA* for type II queries as well.

VI. CONCLUSIONS

In this paper we discussed keyword search over relational databases, in particular the effectiveness aspect of it and how to enhance it. We gave an overview of how the answers to a keyword query are found in a relational database and how they are ranked in the order of their relevance to the query. The first couple of works in this area only consider the size of the tuples when ranking the answers. The more recent works in the area have addressed the shortcomings of the earlier works in terms of effectiveness by leveraging the state-of-the-art IR-style ranking techniques already provided in modern relational database systems and further improved it by few proposed normalizations factors. We identified two other important factors: keyword proximity and keyword quadgrams that should be incorporated into the ranking function. Our experiments show that incorporation of each of these factors improves the effectiveness of the ranking function. We observed that the base algorithm was not able to rank the most relevant answers highest if the given query contained misspellings and especially if the target text attribute includes keyword terms apart from one another.

In our future work we would like to employ more advanced state-of-the-art IR ranking strategies in ranking keyword search results returned from relational databases. Another area, which we are currently investigating and which requires additional research to be done, is the effectiveness of keyword search across heterogeneous relational database.

REFERENCES

[1] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Effcient ir-style keyword search over relational databases. In *VLDB*, pages 850-861, 2003.

[2] V. Hristidis and Y. Papakonstantinou. Discover: Keyword search in relational databases. In *VLDB*, pages 670–681, 2002.

[3] Fang Liu, Clement Yu Weiyi Meng "Effective Keyword search in RDBMS" SIGMOD 2006 Chicago, Illinois

[4] Sanjay Agrawal, Surajit Chaudhuri, Gautam Das, "DBXplorer: A System for Keyword-Based Search over Relational Databases," icde, pp.0005, 18th International Conference on Data Engineering (ICDE'02), 2002

[5] T. Tao and C. Zhai. An exploration of proximity measures in information retrieval. In *SIGIR '07*, pages 295–302, 2007

[6] Luo, Y., Lin, X., Wang, W., and Zhou, X. Spark: top-k keyword query in relational databases. In Proc. SIGMOD (New York, NY, USA, 2007), ACM, pp. 115-126.

[7] R. Goldman, N. Shivakumar, S. Venkatasubramanian and H. Garcia-Molina: Proximity Search in Databases. VLDB, 1998

[8] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan.Keyword searching and browsing in databases using BANKS. In Proc. of ICDE'02, 2002.

[9] IMDB datasets, http://www.imdb.com/interfaces