A Virtual Document Approach for Keyword Search in Databases

Jaime I. Lopez-Veyna, Victor J. Sosa-Sosa and Ivan Lopez-Arevalo
Information Technology Laboratory, Center of Research and Advanced Studies of the National Polytechnic Institute (CINVESTAV), Cd. Victoria, Tamaulipas, Mexico

Keywords: Keyword Search, Indexing, Databases, Top-k, Virtual Documents.

Abstract: It is clear that in recent years the amount of information available in a variety of data sources, like those found on the Web, has presented an accelerated growth. This information can be classified based on its structure in three different forms: unstructured (free text documents), semi-structured (XML documents) and structured (a relational database or XML database). A search technique that has gained wide acceptance for use in massive data sources, such as the Web, is the keyword based search, which is simple to people who are familiar with the use of Web search engines. Keyword search has become an alternative to users without any knowledge about formal query languages and schema used in structured data. There are some traditional approaches to perform keyword search over relational databases such as Steiner Trees, Candidate Networks and recently Tuple Units. Nevertheless these methods have some limitations. In this paper we propose a Virtual Document (VD) approach for keyword search in databases. We represent the structured information as graphs and propose the use of an index that captures the structural relationships of the information. This approach produce fast and accuracy results in search responses. We have conducted extensive experiments on large-scale real databases and the results demonstrates that our approach achieves high search efficiency and high accuracy for keyword search in databases.

1 INTRODUCTION

In recent years we have been witnesses to an exponential increase of the quantity of information available on the World Wide Web. The information search has become an indispensable component in our lives. Web search engines such as Google or Yahoo are the primary way to access massive information. Most of data in the Web can be found in a textual format, and it is also common to find a huge amount of information stored in relational databases, XML documents, and other data sources. This information could be classified based on its structure such as: a) unstructured data, or free-text documents (emails, news, HTML documents) that are written in natural language, b) semi-structured, such as posting on newsgroup (e.g. apartment rentals), medical records, equipment maintenance logs; XML documents are semi-structured because the schema information is mixed with the data values. HTML pages on the Web are considered semi-structured, since the embedded data are often rendered regularly via the use of HTML tags. And c) structured, such as those found in relational databases.

The search engines have used a very simple and widely accepted mechanism for querying textual documents known as keyword search. This mechanism becomes an alternative of querying over relational databases and XML documents, because it is simple to people who are familiar with the use of Web search engines. Recently the database research community has recognized the benefits of keyword search and has been introducing keyword search capability into *relational databases* (Li and Feng, 2009),(Agrawal and Chaudhuri, 2002),(Ding and Xu, 2007),(Hristidis and Papakonstantinou, 2002),(Park and goo Lee, 2011),(Feng and Li, 2011),(Bhalotia and Hulgeri, 2002), *XML databases* (He and Wang, 2007),(V. Hristidis, 2006),(Hristidis and Papakonstantinou, 2003), (Bao and Lu, 2010), *graph databases* (Kimelfeld and Sagiv, 2008),(Achiezra and Golenberg, 2010),(Kacholia and Pandit, 2005),(Zhong and Liu, 2009),(He and Wang, 2007), and *heterogeneous data sources* (Dong and Halevy, 2007),(Li and Feng, 2008a),(Franklin and Halevy, 2005). One important advantage of keyword search is that it enables users to search for information without having a detailed knowledge of the schema of the database or XML document and without the need to learn some formal languages like SQL or XQuery.

Although keyword search has been proven to be effective for textual documents, it presents some limitations on structured and semi-structured data that are not easy to carry out. This situation is because current Information Retrieval (IR) techniques applied on search engines have not been designed for this type of data sources (M. Karnstedt, 2008). They have always employed the inverted index to process keyword queries, which is effective for unstructured data but it is inefficient for semi-structured and structured data (Franklin and Halevy, 2005). Additionally, these techniques ignore the information structure (hyperlinks in the case of unstructured data, parent child relationships or IDRefs in semi-structured data and primary and foreign keys in the case of structured data) that can be extracted from the data sources for answering keyword queries.

We propose a novel virtual document approach as an attempt to solve the problem of keyword search in databases. This approach can be applied to unstructured and semi-structured information. Our approach will improve the keyword search in databases and make an efficient processing using a score based on classical information retrieval that takes into account the meta-information included in structured information.

The rest of this paper is organized as follows. Section 2 shows the Related Work. We introduce the Problem in Section 3. Section 4 presents our approach. Section 5 shows some of our preliminary results. Conclusions and Future Work are given in section 6.

2 RELATED WORK

The first area of research related to our work is the keyword-based search over structured data using the IR techniques. Inverted index has been proven to be one of the most effective techniques to organize unstructured data. In recent years there are applications that need to integrate structured data and text documents. Due to this situation, different approaches to keyword search over structured data have been proposed. The existing approaches can be classified into three main methods: *Steiner Tree*, *Candidate Networks* and *Tuple Units*.

The Steiner-Tree-based methods use a data graph structure to compute relevant answers. A Steiner Tree can be defined as: given N points in a plane, it is necessary to connect them by lines of minimum total length in such a way that any two points may be interconnected(Du and Hu, 2008). The authors in (Li and Feng, 2009),(He and Wang, 2007),(Li and Feng, 2008a),(Dong and Halevy, 2007) and (Bhalotia and Hulgeri, 2002) first model the tuples in a relational database as a data graph, where nodes are tuples and edges are the primary-foreign-key relationships, and they identify the steiner trees that contain all or some of the input keywords to answer a query. These methods compute answers on the fly by traversing the database graph to discover structural relationships. Since minimum Steiner tree problem is known to be NP-hard (Fang and Clement, 2006), a Steiner Tree based approach can be inefficient, demanding new studies to find polynomial time solutions that can effectively approximate the minimum Steiner Tree problem (Li and Feng, 2009). For example, the EASE approach proposed by (Li and Feng, 2008a) was reviewed and we found that the datagraph generated from the IMDB Dataset[1], contained 1,000,209 nodes and 200,000 vertex. After following the process described in this approach, we found 1,007,826 minimal Steiner Trees, this is a big number, closer to the same number of nodes of the datagraph, requiring an important computational effort. Another issue found in the EASE approach is the need for computing a score DB, which requires to analyze all possible paths between two nodes belonging to a Steiner Tree. These issues imply a very high computational effort and time consuming process, especially in the cases where a Minimal Steiner Graph contains hundred of nodes and arcs. BLINKS(He and Wang, 2007), BANKS(Bhalotia and Hulgeri, 2002), EASE(Li and Feng, 2008a) and CSTREE(Li and Feng, 2009) are examples of this approach.

The Candidate Networks based methods use the database or XML schema to identify answers. A Candidate Network is a subtree that involves tuple sets with occurrences of the query keywords, where the keywords can be leafs nodes or the root of the subtree. They first generate candidate networks following the primary-foreign-key relationships, and then compute answers composed of relevant tuples based on Candidate Networks. Each Candidate Network would be generated and evaluated as a good answer. The problem of evaluating all Candidate Networks is a multi-query optimization problem and has two main issues: (1) How to share common subexpressions among Candidate Networks generated in order to reduce computational cost evaluation and (2) How to find a proper join order to fast evaluate all Candidate Networks(Xu and Qui, 2010), this is because for a keyword query, the number of Candidate Networks can be very large. DISCOVER(Hristidis and Papakonstantinou, 2002) and SPARK(Luo and Lin, 2007) are examples of this approach.

[1]http://www.imdb.com/interfaces (1,000,000 Data Set (475 MB)).

These first two methods discover the connections (primary/foreign keys) between tuples on the fly. As there are usually large numbers of tuples in a database, these methods are rather expensive to find answers and neglect that relevant tuples can be identified, pre-computed, materialized and indexed offline(Li and Feng, 2008b).

The Tuple Units are proposed by Li et. al. in (Li and Feng, 2008b). A Tuple Unit is composed of relevant tuples connected by primary-foreign-key relationships. It is a set of highly relevant tuples which contain query keywords. The tuple units are generated and materialized off-line in order to accelerate the on-line processing of keyword queries. EKSO(Su and Widom, 2005) and RETUNE(Li and Feng, 2008b) are examples of this approach. Following the RETUNE approach proposed by (Li and Feng, 2008a). We tried to make our own implementation in order to make a real comparison with this approach. Unfortunately, it was not possible to create the score table that contains the posting list that the authors created for a single keywords. They mentioned the use of MYSQL in this process. However MYSQL is limited to 4096 columns per table or every table has a maximum row size of 65,535 bytes[2]. In this process, at leas 4170 columns to store terms and 126,505 columns to store keyword pairs were required. This situation made impossible to reproduce those experiments.

3 PROBLEM

3.1 Motivation

Nowadays information becomes the most critical and valuable asset for business. Providing a uniform interface for querying and retrieving information regardless of its structure, is not a trivial task. Having access to a large set of structured data or heterogeneous data has been one of the most important challenges facing the current database and information recovery/retrieval projects (Li and Feng, 2008a),(Chaudhuri and Ramakrishnan, 2005),(Abiteboul and Allard, 2008). It Is very common to find information on the web by typing some keywords in a search engine (using IR techniques). However, some times the returning results may not contain relevant data. Current search engines are mainly focused on exploring unstructured information (documents). In this way, they are losing relevant information that could be found in different data sources like semi-structured or structured. To have access to structured or semistructured data, users need to know the schema of the database or to learn some formal language like SQL, XQuery (Li and Feng, 2009). Furthermore, the IR-style ranking model implemented in current search engines ignores the information structure (meta-information) that can be extracted from different data sources. This meta-information can be useful for determining the relevance of that information(Li and Feng, 2008a). The structure information awareness facilitates the assembling of pieces of data that are interconnected at different relations or XML documents. These pieces of data can represent more relevant answers that could be provided by a search engine.

The mentioned situation has motivated us to propose a novel *Virtual Document Approach for Keyword Search in Databases*. This approach makes an efficient query processing and uses a score based on the classical Information Retrieval metrics. Furthermore it takes into account the structure extracted from the structured information. Our approach identifies the most relevant and meaningful tuples to answer keyword queries. We model structured data as graphs, where nodes can be tuples, and edges can be primary-foreign-key relationships. A data index enables efficient keyword search over databases using an IR style that captures the structural relationships of the data. This structure is kept implicitly in what is called Virtual Documents..

The main contributions of this paper are:

- A novel and efficient keyword search method for structured information.
- A new method for the construction of virtual documents from structured data.
- A definition of a data index that stores the structural relationships taken from the data sources.
- The analysis of the issues related to indexing and ranking, and a simple and efficient indexing mechanism to index the structural relationships between the transformed data.
- The development of an effective ranking technique to rank the virtual documents by taking into account both the structural relationships included in the relational and XML databases and the textual relevancy using a text retrieval approach.

4 OUR APPROACH

Our Virtual Document Approach is a novel approach for keyword search on databases that takes some ideas

[2]http://dev.mysql.com/doc/refman/5.0/en/column-count-limit.html

from the RETUNE project (Li and Feng, 2008b) but implementing important differences. The first difference is that we manipulated the information in a data graph, instead of using relational tables. The second difference is that RETUNE uses SQL to extract the tuple units and our approach uses the data graph to identify the Tuple Units that are converted into something called Virtual Documents. These documents are used for creating a data index that takes into account the meta-information included in structured data without including redundant information.

4.1 Notations

This section introduces some notations for clarity of this paper. In our approach we model structured data as an undirected graph, where the nodes are tuples, and the edges are primary-foreign-key relationships. A data Graph can be defined as follows:

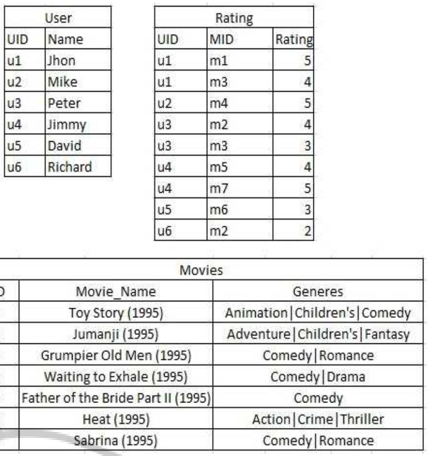
Definition 1. *(Data Graph) A Data Graph G is a pair* (V,E)*, where V is a set of vertices, and E is a set of edges between the vertices* $E \subseteq \{(u,v)|u,v \in V\}$

We need a database to identify Tuple Units and transform them into Virtual Documents. Given a database D with n tables, $R_1, R_2 \ldots, R_n$, $R_i \underset{\rightarrow}{k} R_j$ denote that R_i has a foreign key k which refers to the primary key of R_j. If two relational tables R_i and R_j are connected, are denoted as, $R_i \Leftrightarrow R_j$, if *i)* $R_i \underset{\rightarrow}{k} R_j$; or *ii)* $R_j \underset{\rightarrow}{k} R_i$; or *iii)* $\exists R_k, R_i \Leftrightarrow R_k$ and $R_k \Leftrightarrow R_j$. In this approach, we suppose that each table is pointed out by other relations that act as link table. A relational table R_i is called a *link table* if there is not relation table R_j, such that, $R_j \underset{\rightarrow}{k} R_i$. That is, R_i only contains foreign keys to connect other relational tables but it does not contain any primary key (Li and Feng, 2008b). For example, consider the database with three tables in the figure 1, we have $Rating \underset{\longrightarrow}{UID} User$ and $Rating \underset{\longrightarrow}{MID} Movie$; and $User \leftarrow Rating \rightarrow Movie$. Rating is a link relational table as it has no primary key.

4.2 Tuple Units

Since the database is modeled as a data graph, we can identify a Tuple Unit following the primary and foreign keys into the data graph. We have to choose a root node of the data graph and the set of neighbors or adjacent nodes connected by primary or foreign key, will be called a Tuple Unit. Next this Tuple Unit is transformed in a set of Virtual Documents. Our version of Tuple Units is defined as follows:

Definition 2. *(Tuple Unit (TU)) Given a node* $v_i \in V$ *a Tuple Unit from the node* v_i*, are composed of the set of neighbors of the node* v_i*, and can be defined as* $TU(v_i) = \{u|u \in neighbors(v_i)$ *and* $u \in V\}$*. The node* $TU(v_i)$ *is called the root node from the Tuple Unit of* (v_i).



User	
UID	Name
u1	Jhon
u2	Mike
u3	Peter
u4	Jimmy
u5	David
u6	Richard

Rating		
UID	MID	Rating
u1	m1	5
u1	m3	4
u2	m4	5
u3	m2	4
u3	m3	3
u4	m5	4
u4	m7	5
u5	m6	3
u6	m2	2

Movies		
MID	Movie_Name	Generes
m1	Toy Story (1995)	Animation|Children's|Comedy
m2	Jumanji (1995)	Adventure|Children's|Fantasy
m3	Grumpier Old Men (1995)	Comedy|Romance
m4	Waiting to Exhale (1995)	Comedy|Drama
m5	Father of the Bride Part II (1995)	Comedy
m6	Heat (1995)	Action|Crime|Thriller
m7	Sabrina (1995)	Comedy|Romance

Figure 1: An example database.

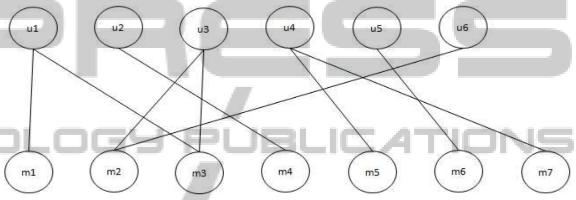
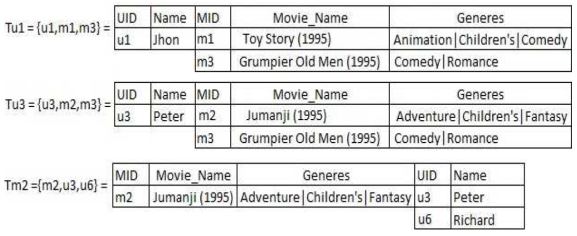


Figure 2: The graph model for the example database in Figure 1.

To better understand the concept of tuple *Tuple Units (TU)*, we give the Example 1.

Example 1. *Consider the database in the figure 1 where Rating is a link table. First we model as a data graph following the primary and foreign keys as illustrated in figure 2. With this graph we can iteratively get each node in the graph and obtain the TU. For example, consider the nodes u1, u3 and m2. We can get the Tuple Units Tu1, Tu3 and Tm2 respectively, as shown in figure 3*

The Tuple Unit based method has the following features: (1) TU is effective to answer keyword



Tu1 = {u1,m1,m3} =

UID	Name	MID	Movie_Name	Generes
u1	Jhon	m1	Toy Story (1995)	Animation|Children's|Comedy
		m3	Grumpier Old Men (1995)	Comedy|Romance

Tu3 = {u3,m2,m3} =

UID	Name	MID	Movie_Name	Generes
u3	Peter	m2	Jumanji (1995)	Adventure|Children's|Fantasy
		m3	Grumpier Old Men (1995)	Comedy|Romance

Tm2 ={m2,u3,u6} =

MID	Movie_Name	Generes	UID	Name
m2	Jumanji (1995)	Adventure|Children's|Fantasy	u3	Peter
			u6	Richard

Figure 3: Tuple units from the nodes u1, u3 and m2.

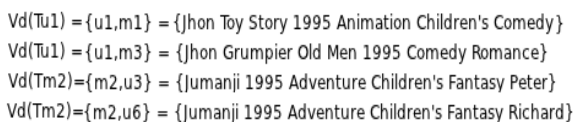

Vd(Tu1) ={u1,m1} = {Jhon Toy Story 1995 Animation Children's Comedy}
Vd(Tu1) ={u1,m3} = {Jhon Grumpier Old Men 1995 Comedy Romance}
Vd(Tm2)={m2,u3} = {Jumanji 1995 Adventure Children's Fantasy Peter}
Vd(Tm2)={m2,u6} = {Jumanji 1995 Adventure Children's Fantasy Richard}

Figure 4: Virtual documents from the nodes u1 and m2.

queries as they capture structures and can represent a meaningful and integral information unit. (2) The relationships between tuples connected through primary/foreign keys can be identified and indexed, so we can efficiently answer keyword queries by using such indexed structural information. (3) The number of TU will not be large, which is not larger than of the total tuples in the underlying database(Ding and Xu, 2007).

Once we identify the TU for each node in the graph, we need to transform it into a Virtual Document by dividing the TU into a set of documents. We indeed create a new document for each neighbors of the node, and all these documents together compose a single Virtual Document. The new Virtual Document only contains the textual attributes from TU, the idea is to reduce the size of virtual documents by avoiding redundant information. We can also include non textual attributes, such as numbers and dates, taking their string representation. A similar approach to virtual documents was proposed by (Su and Widom, 2005). The set of virtual documents generated will feed the index. The figure 4 shows the virtual documents generated from the Tuple Units Tu1 and Tm2.

4.3 Ranking

Section 4.2 described how Tuple Units were identified and transformed into Virtual Documents(VD). In this section, we first discuss how to meaningfully index and rank the Virtual Documents and then to identify the top-k answer based on existing proposals. The generated VDs include the structural relationships between input keywords with respect to a Tuple Unit.

The simplest way to score and rank the virtual documents is by the use of the TF · IDF metric. We can take the terms in the virtual documents as keywords, following a similar technique of indexing and ranking used in DISCOVER2 (Hristidis and Gravano, 2003), SPARK (Luo and Lin, 2007), EASE (Li and Feng, 2008a) and by (Fang and Clement, 2006), where the terms can be used to answer keyword-based queries over Virtual Documents. The idea is first assigning to each Virtual Document a score using a standard IR-ranking formula, and then to combine the individual scores using a score aggregation function, such as SUM, to obtain the final score.

Assuming that we have calculated the total set of the virtual documents denoted as D, there are N different virtual documents and K keywords in D. Given a virtual document denoted as $d \in D$ and a keyword k_i contained in d. We set $ntf(k_i,d)$ as the normalized term frequency of k_i in a document d, and it is defined in Equation 1. Where $tf(k_i,d)$ represents the number of occurrences of the term k_i in a document d.

$$ntf(k_i) = 1 + ln(1 + tf(k_i,d)) \tag{1}$$

We set $idf(k_i)$ as the inverse document frequency of k_i defined in equation 2. Where df_{ki} is the number of virtual documents that a term k_i occurs in D. Idf is normalized by dividing the total number of Virtual Documents N over $(df_{ki}+1)$ and then applying the ln function.

$$idf(k_i) = ln\frac{N}{df_{ki}+1} \tag{2}$$

Document Normalized Length $ndl(d)$ is defined in Equation 3, and represents the normalized document length, that is the number of terms in the document d over the average of terms in the set of documents D. ndl is used to reduce the term weights in long documents. $|d|$ denotes the number of terms in a document d. And s is a constant taken from IR literature(Fang and Clement, 2006) and is usually set to 0.2.

$$ndl(d) = (1-s) + s * \frac{|d|}{\frac{\sum_{d' \in D}|D'|}{N}} \tag{3}$$

In the IR literature, ranking methods usually combine the three metrics (tf, idf and ndl) in a $TF \cdot IDF$ for a keyword k_i in a Virtual Document d as illustrated in equations 4.

$$TF \cdot IDF(k_i,d) = \frac{ntf(k_i)}{ndl(d)} * idf(k_i) \tag{4}$$

Although the TF · IDF-based ranking methods are efficient for textual documents, they are inefficient for semi-structured and structured data and do not take into account the implicit structured information. In our approach, Virtual Documents capture some structural information, which is easy to find in relational databases and XML documents. Representing the rich structural relationships by the Virtual Documents should be at least as important as discovering more keywords, and in some cases, even more crucial.

Therefore, given a keyword query $K = k_1, k_2, ...k_n$, and a Virtual Document d we can compute the score of a virtual document d with respect to a keyword query K by summing the TF · IDF for each keyword $k_i \in K$ as it is defined in equation 5.

$$SCORE(K,d) = \sum_{k=1}^{n} TF \cdot IDF(k_i,d) \tag{5}$$

With the compute of the *SCORE*, we are allowed to answer queries and rank the relevant virtual documents, taking into consideration the classical Information Retrieval metrics and including in this score the structural information extracted from the relationships of a Tuple Unit.

4.4 Indexing

To efficiently retrieve scores of the documents, we propose a virtual document inverted index *(VDII)*, which is similar to the traditional inverted index. The entries of VDII are also the keywords that are contained in the virtual documents. VDII is similar to inverted indexes since each entry k_i keeps the virtual documents that directly contain the keyword, and the corresponding score. The virtual documents with respect to the entry k_i are sorted by $SCORE(k_i,d)$ in descending order, where $d \in \{d_j | d_j$ contains the keyword $k_i\}$. We indexing all the virtual documents using the Apache Hadoop Framework [3]. Hadoop is an open source framework for writing and running distributed applications that process large amounts of data using commodity hardware(Lam, 2011). Hadoop uses an implementation of Map-Reduce programming model.

4.5 Query Processing

With the *VDII* index, we can answer a keyword query as follows: given a keyword query $K = k_1, k_2, ...k_n$, we first retrieve the inverted lists of $I_i (1 \leq i \leq n)$, which is composed of the virtual documents that contain keyword k_i. Then, we compute the score of each relevant virtual document adding the *SCORE* of each keyword, next we apply a ranking function to finally return the $top - k$ answers with the highest scores.

5 RESULTS

5.1 Experimental Scenario

In order to evaluate the effectiveness and the efficiency of our proposal, we have conducted extensive experiments on real datasets. We compared search efficiency and results quality with existing state of the art algorithms BLINKS(He and Wang, 2007), EASE(Li and Feng, 2008a) and RETUNE(Li and Feng, 2008b). We used the Internet Movie DataBase (IMDB) to evaluate our approach. IMDB contains approximately one million anonymous ratings of 3883 movies made by 6040 users, each user has rated at least 20 movies. IMDB dataset are translated into four tables as illustrated in Table 1.

[3]http://hadoop.apache.org/

Table 1: IMDB dataset.

Tables	Attributes	No. of Records
User	UID, UserName, Gender, OID	6,040
Movie	MID, title, genres	3,883
Occupation	OID, Occupation	30
Rating	UID, MID, Score	1,000,209

We model the data set as undirected graph. This dataset generate about 1,010,153 vertex and 2,006,458 arcs. The elapsed time to indexing IMDB and build the *VDII* index is 741 seconds approximately and the size of the index is close to 355 MB. The *VDII* index includes 4,170 different keywords of the dataset (without stop words) appearing in at least one Virtual Document and 13,766,755 postings, with an average of 3301 posting per keyword, and a maximal of 713,160 postings for a keyword. The elapsed time required to load the index in memory and start to query it requires only 108 seconds approximately.

All the algorithms were coded in Java. All the experiments were conducted on a computer with an Intel(R) Core (TM) 2@2.33 GHz, CPU, 4 GM RAM running Ubuntu 11.10.

5.2 Search Efficiency

This section evaluates the search efficiency of various algorithms. We selected one hundred keyword queries with several number of keywords (2 to 6). To form a query, we first selected a random number of keywords from the movie title, next we selected a random number of keywords from the Movie's genre. Finally, we selected a random number of keywords from occupation of the evaluator of movies. We group the queries that have the same number of keywords and then compute the average of the elapsed time. Table 2 shows various sample queries. The figure 5 illustrates the elapsed time from the queries. We can note that the average of the elapsed time increases with respect to the value of the number of keywords. This is because we need to find more virtual documents to calculate the final score. For example, in a query with four keywords we get 132 ms to identify the $top - 100$ results and compute the final score. We can also observe that our algorithm produces better results than the existing methods BLINKS, EASE and RETUNE. VDII clearly reach high efficiency in the search, this is because our approach does not need to identify answers by discovering the relationships between tuples that appears in different relational tables on the fly, or

Table 2: Several sample queries employed in the experiments.

Examples of queries
herbie bananas
super mario 1993
toy story 2 animation
back future 1990 western writer
star wars iv hope adventure retired

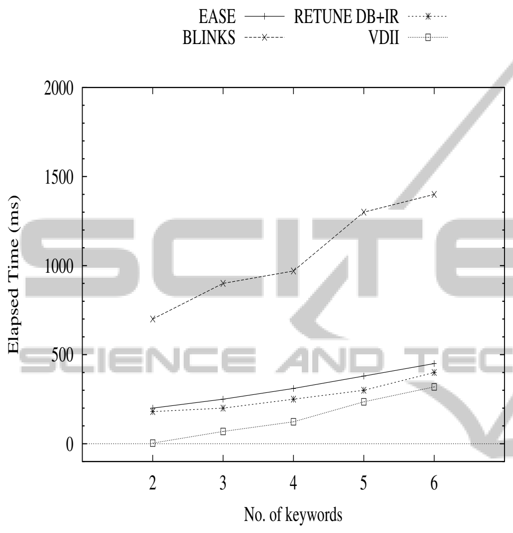


Figure 5: Search efficiency using 2 to 6 keywords.

using SQL, since we include that relationships in the virtual documents in the VDII index. VDII is faster than EASE, this is because EASE spend some time extracting the Steiner Graphs and eliminating the non-Steiner nodes. The elapsed time of BLINKS increases considerably when adding more keywords, compared with the elapsed time of VDII that varies a few ms, achieving higher search efficiency. This difference is due to the use of VDII index to identify the answers in our approach. VDII takes less than 320 ms to answer keyword queries with six keywords; EASE takes 420 ms and BLINKS uses more than 1,200 ms. Clearly VDII outperforms BLINKS and is also significantly faster than EASE.

To better evaluate performance of our approach, we identified the top-k answers with different values of k and compared the corresponding elapsed time. The figures 6 and 7 show the experimental results. We can see that our approach outperforms the existing methods BLINKS and EASE significantly. For example in the Figure 6 VDII takes 121 ms to identify the top 10 answers, while BLINKS and EASE take more than 200 ms and 1400 ms respectively. Furthermore, with the increase of the number of $top-k$ answers the elapsed time of VDII varies slightly and always

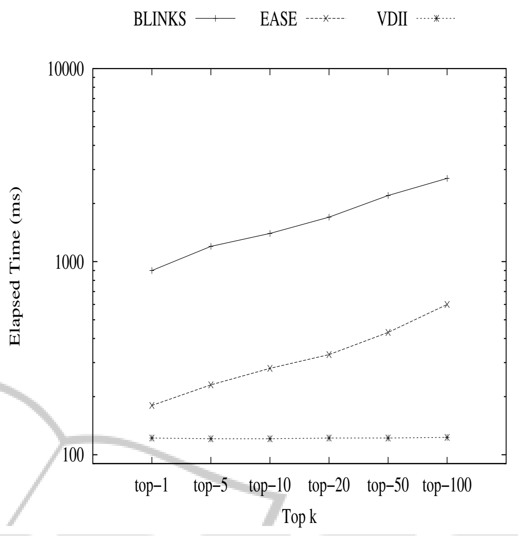


Figure 6: Top-k search efficiency using 4 keywords.

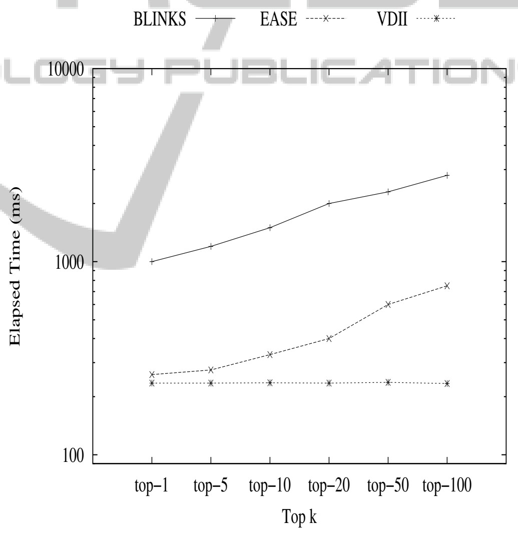


Figure 7: Top-k search efficiency using 5 keywords.

achieves high efficiency on the search.

5.3 Search Accuracy

This section evaluate the search accuracy using standard precision. The lack of a benchmark that allows us to verify the precision of the obtained results for a set of queries motivated us to define a basic strategy for measuring the accuracy. Document relevancy is measured by using two constraints: First, a document is relevant if the query keywords appear in some of its textual attributes. Second, it is also relevant if these keywords are located in a semantically related attribute. This is not an automatic validation, for example, if we wanted to find the rates given by

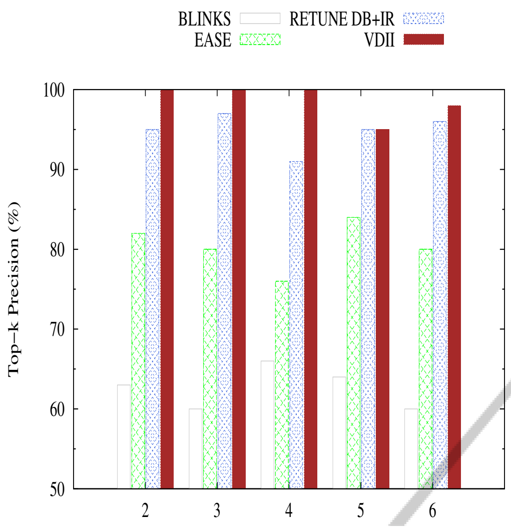


Figure 8: Top-k search precision.

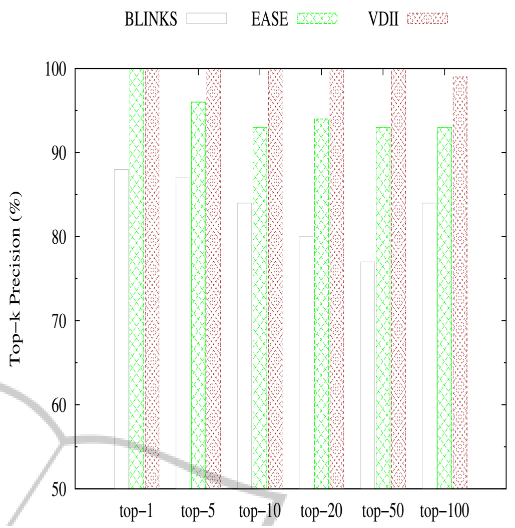


Figure 9: Top-k search precision using 4 keywords.

Writers to the movie Doctor Dolittle (keywords: Doctor Dolittle Writer), a document would be relevant if the keywords Doctor Dolittle are appearing in the movie name attribute and the keyword Writer appears in the occupation attribute. In this case, it is easy to see that false positive documents could be those including movies that were rated by Doctors. So far, this validation is carried out manually. We also used the metric top-k precision, which measures the ratio of the number of relevant answers among the first k answers with the highest scores. Figure 8 shows the experimental results. In this figure we can observe that our approach obtains higher search accuracy and outperforms the existing methods. VDII is better than BLINKS in various queries with different number of keywords. This is because our approach includes structural information in the index construction when applying the TF · IDF based IR technique. VDII also outperforms EASE and RETUNE in terms of precision.

Figures 9 and 10 illustrate the experimental results, which show that VDII achieves higher precision than BLINKS and EASE on different values of k. We can see that with the increase in the k values, the $top-k$ precision of BLINKS falls while VDII can always achieve high precision. This confirm the efficiency of our method.

We need to make other tests with VDII to compare the results with other approaches, however, we can observe that VDII achieves high search efficiency and quality for keyword search.

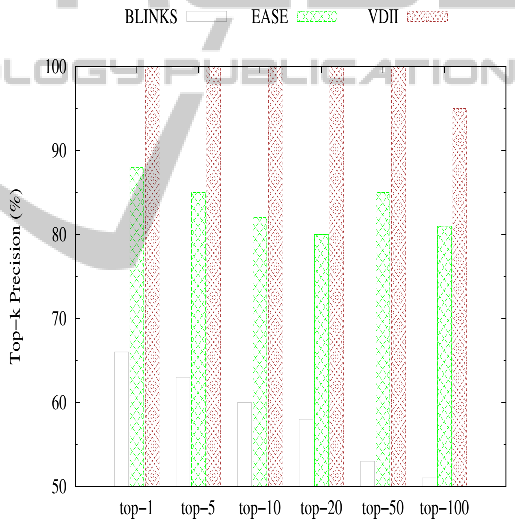


Figure 10: Top-k search precision using 5 keywords.

6 CONCLUSIONS AND FUTURE WORK

Searching information has become an indispensable component in our lives. There is a need to find information in data sources with heterogeneous structure including databases. In this paper, we have studied the problem of effective keyword search over structured data and present a *Virtual Document Approach for Keyword Search in Databases* to make an efficient processing of keyword queries. Our approach is based on the classical Information Retrieval metrics taking into account the meta-information extracted from structured information. We model structured data as graph and have proposed the use of a data index *(VDII)* to capture the structural relationships for fast and accuracy response. Finally, we have conducted some experiments to evaluate the efficiency and effectiveness of our approach using real data sets. This experiments show that our approach achieves high search efficiency and quality for keyword search and is capable to scale with databases with tens of millions of tuples.

Our future work includes the need to continue the testing of our approach with other datasets including semi-structurated and unstructured data. We also have to continue working on some strategies to reduce of the size of the index for example by the using of Mutual Information.

ACKNOWLEDGEMENTS

This research was partially supported by project number 187325 from Fondo Mixto Conacyt-Gobierno del Estado de Tamaulipas.

REFERENCES

Abiteboul, S. and Allard, T. (2008). Webcontent: Efficient p2p warehousing of web data.

Achiezra, H. and Golenberg, K. (2010). Exploratory keyword search on data graphs. In *Proceedings of the 2010 international conference on Management of data (SIGMOD)*, pages 1163–1166. ACM.

Agrawal, S. and Chaudhuri, S. (2002). Dbxplorer: A system for keyword-based search over relational databases. In *Proceedings of the 18th International Conference on Data Engineering*, ICDE ’02. IEEE Computer Society.

Bao, Z. and Lu, J. (2010). Towards an effective xml keyword search. *IEEE Transactions on Knowledge and Data Engineering*, 22(8):1077–1092.

Bhalotia, G. and Hulgeri, A. (2002). Keyword searching and browsing in databases using banks. In *Proceedings of the 18th International Conference on Data Engineering*, ICDE ’02, pages 431–440.

Chaudhuri, S. and Ramakrishnan, R. (2005). Integrating db and ir technologies: What is the sound of one hand clapping. In *Innovative Data Systems Research (CIDR)*, pages 1–12.

Ding, B. and Xu, J. (2007). Finding top-k min-cost connected trees in databases.

Dong, X. and Halevy, A. (2007). Indexing dataspaces. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, SIGMOD ’07, pages 43–54. ACM.

Du, D. and Hu, X. (2008). *Steiner Tree problems in Computer Communication Networks*. World Scientific Publishing.

Fang, L. and Clement, Y. (2006). Effective keyword search in relational databases. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, SIGMOD ’06, pages 563–574. ACM.

Feng, J. and Li, G. (2011). Finding top-k answers in keyword search over relational databases using tuple units. *IEEE Transactions on Knowledge and Data Engineering Volume*, 23:1781–1794.

Franklin, M. and Halevy, A. (2005). From databases to dataspaces: A new abstraction for information management. *SIGMOD Record*, 34:27–33.

He, H. and Wang, H. (2007). Blinks: ranked keyword searches on graphs. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, SIGMOD ’07, pages 305–316. ACM.

Hristidis, V. and Gravano, L. (2003). Efficient ir-style keyword search over relational databases. In *Proceedings of the 29th international conference on Very large data bases - Volume 29*, VLDB ’2003, pages 850–861. VLDB Endowment.

Hristidis, V. and Papakonstantinou, Y. (2002). Discover: Keyword search in relational databases. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 670–681. VLDB Endowment.

Hristidis, V. and Papakonstantinou, Y. (2003). Keyword proximity search on xml graphs. In *Proceedings. 19th International Conference Data Engineering*, pages 367–378.

Kacholia, V. and Pandit, S. (2005). Bidirectional expansion for keyword search on graph databases.

Kimelfeld, B. and Sagiv, Y. (2008). Efficiently enumerating results of keyword search over data graphs. *Information Systems*, 33:335–359.

Lam, C. (2011). *Hadoop in Action*. Manning Publications Co.

Li, G. and Feng, J. (2008a). Ease: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data(SIGMOD)*, pages 903–914.

Li, G. and Feng, J. (2008b). Retrieving and materializing tuple units for effective keyword search over relational databases. In *Lecture Notes in Computer Science, Conceptual Modeling - ER*, pages 469–483.

Li, G. and Feng, J. (2009). Providing built-in keyword search capabilities in rdbms.

Luo, L. and Lin, X. (2007). Spark: top-k keyword query in relational databases. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, SIGMOD ’07, pages 115–126. ACM.

M. Karnstedt, K. S. (2008). A dht-based infrastructure for ad-hoc integration and querying of semantic data. In *Proceedings of the 2008 international symposium on Database engineering and applications*, pages 19–28.

Park, J. and goo Lee, S. (2011). Keyword search in relational databases. *Knowl. Inf. Syst*, 26(2):175–193.

Su, Q. and Widom, J. (2005). Indexing relational database content offline for efficient keyword-based search. In

Proceedings of the 9th International Database Engineering and Application Symposium (IDEAS), pages 297–306.

V. Hristidis, N. K. (2006). Keyword proximity search in xml trees. *IEEE Transactions on Knowledge and Data Engineering*, pages 525–539.

Xu, J. and Qui, L. (2010). Keyword search in relational databases: A survey. *Bulletin of the IEEE Computer Society Technical Comittee on Data Engineering*, 33:67–78.

Zhong, M. and Liu, M. (2009). Efficient keyword proximity search using a frontier-reduce strategy based on d-distance graph index. In *Proceedings of the 2009 International Database Engineering & Applications Symposium (IDEAS)*, pages 206–216. ACM.