

# KESOSD: Keyword Search Over Structured Data

Jaime I. Lopez-Veyna Victor J. Sosa-Sosa Ivan Lopez-Arevalo  
Information Technology Laboratory  
Center of Research and Advanced Studies of  
the National Polytechnic Institute (CINVESTAV)  
Cd. Victoria, Tamaulipas, Mexico  
(52) 834 107 0220 + 1114, 834 107 0241, 834 107 0220  
{jlopez,vjsosa,ilopez}@tamps.cinvestav.mx

## ABSTRACT

Most of the information on the Web can be currently classified according to its (information) structure in three different forms: unstructured (plain text), semi-structured (XML files) and structured (tables in a relational database). Currently Web search is the primary way to access massive information. Keyword search also becomes an alternative of querying over relational databases and XML documents, which is simple to people who are familiar with the use of Web search engines. There are several approaches to perform keyword search over relational databases such as Steiner Trees, Candidate Networks and Tuple Units. However these methods have some constraints. The Steiner Trees method is considered a NP-hard problem, moreover, a real databases can produce a large number of Steiner Trees, which are difficult to identify and index. The Candidate Network approach first needs to generate the candidate networks and then to evaluate them to find the best answer. The problem is that for a keyword query the number of Candidate Networks can be very large and to find a common join expression to evaluate all the candidate networks could require a big computational effort. Finally, the use of Tuple Units in a general conception produce very large structures that most of the time store redundant information. To address this problem we propose a novel approach for keywords search over structured data (*KESOSD*). *KESOSD* models the structured information as graphs and proposed the use of a *keyword-structure-aware-index* called *KSAI* that captures the implicit structural relationships of the information producing fast and accuracy search responses. We have conducted some experiments and the results show that *KESOSD* achieves high search efficiency and high accuracy for keyword search over structured data.

## Categories and Subject Descriptors

H.2.8 [Information Systems]: Database Applications; H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval

*Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.*

KEYS'12, May 20, 2012, Scottsdale, Arizona, USA.

Copyright 2012 ACM 978-1-4503-1198-4/12/06 \$10.00.

## General Terms

Algorithms, Experimentation, Performance

## Keywords

keyword search, indexing, databases, top-k, virtual documents, graphs

## 1. INTRODUCTION

In recent years we have been witnesses to an exponential increase of the quantity of information available on the World Wide Web (7.09 billion pages)<sup>1</sup>. 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. The origin of the information comes from a variety of data sources that 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 [21, 3, 7, 14, 25, 10, 5], XML databases [12, 27, 15, 4], graph databases, [17, 2, 16, 29, 12], and heterogeneous data sources [8, 19, 11]. 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.

<sup>1</sup>The size of the Indexed Web (visited on February 27th ,2012, <http://www.worldwidewebsize.com/>)

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 [24]. 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 [11]. 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.

To address the problem of Keyword-based Search Over Structured Data, we propose a novel approach called KESOSD as an attempt to solve these problems. This approach can be applied to unstructured and semi-structured information. KESOSD will improve the keyword-based search over structured data 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. We introduce the Problem in Section 2. Section 3 presents KESOSD. Section 4 shows some of our preliminary results. Related work, conclusions and Future Work are given in sections 5 and 6 respectively.

## 2. PROBLEM

### 2.1 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 [19, 6, 1]. It is very common to find information on the web by using some keywords to 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 [21]. 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 [19]. 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 approach for *Keyword Search Over Structured Data (KESOSD)*. KESOSD makes an efficient processing querying

and uses a score based on the classical Information Retrieval metrics. Furthermore it takes into account the structure extracted from the structured information. KESOSD identifies the most relevant and meaningful tuples to answer keyword queries. In our approach, we model structured data as graphs, where nodes can be tuples, and edges can be primary-foreign-key relationships. We enable efficient keyword search over structured data sources by the use of a data index, *keyword-structure-aware-index* called KSAI. It uses an IR style that captures the structural relationships of the data that will be represented in something 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 *keyword-structure-aware-index (KSAI)* that store 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.

## 3. OUR APPROACH: KESOSD

KESOSD is a novel approach for keyword-search on structured data that takes some ideas from the RETUNE project [20] but implementing important differences. The first difference is that we accept the idea of modeling the data sources as a data graph, where the nodes are tuples and edges are the primary-foreign-key relationships. The second difference is that RETUNE uses SQL to extract the tuple units and KESOSD 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 called *keyword-structure-aware-index (called KSAI)* that takes into account the meta-information included in structured data without including redundant information.

### 3.1 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  $E \subseteq \{(u, v) | u, v \in V\}$

User		Rating		
UID	Name	UID	MID	Rating
u1	Jhon	u1	m1	5
u2	Mike	u1	m3	4
u3	Peter	u2	m4	5
u4	Jimmy	u3	m2	4
u5	David	u3	m3	3
u6	Richard	u4	m5	4
		u4	m7	5
		u5	m6	3
		u6	m2	2

Movies		
MID	Movie_Name	Genres
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

We need a database to identify Tuple Units and transform them into Virtual Documents. Given a database  $D$  with  $n$  tables,  $R_1, R_2, \dots, R_n$ ,  $R_i \xrightarrow{k} R_j$  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*)  $R_i \xrightarrow{k} R_j$ ; or *ii*)  $R_j \xrightarrow{k} R_i$ ; 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,  $R_j \xrightarrow{k} R_i$ . That is,  $R_i$  only contains foreign keys to connect other relational tables but it does not contain any primary key [20]. For example, consider the database with three tables in the figure 1, we have  $Rating \xrightarrow{UID} User$  and  $Rating \xrightarrow{MID} Movie$ ; and  $User \leftarrow Rating \rightarrow Movie$ . Rating is a link relational table as it has no primary key.

### 3.2 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  $TU(v_i) = \{u | u \in neighbors(v_i) \text{ and } u \in V\}$ . The node  $TU(v_i)$  is called the root node from the Tuple Unit of  $(v_i)$ .

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

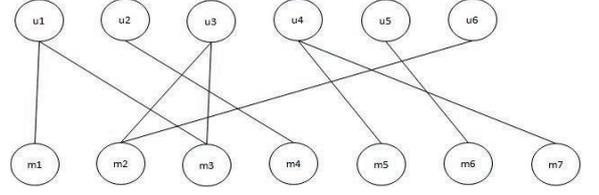


Figure 2: The graph model for the example database in Figure 1

Tu1 = {u1,m1,m3} =				
UID	Name	MID	Movie_Name	Genres
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	Genres
u3	Peter	m2	Jumanji (1995)	Adventure Children's Fantasy
		m3	Grumpier Old Men (1995)	Comedy Romance

Tm2 = {m2,u3,u6} =				
MID	Movie_Name	Genres	UID	Name
m2	Jumanji (1995)	Adventure Children's Fantasy	u3	Peter
			u6	Richard

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

$u1, u3$  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 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 [7].

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 neighbor 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 [26]. The set of virtual documents generated will feed the index. The figure 4 shows the virtual documents generated from the Tuple Units  $Tu1$  and  $Tm2$ .

### 3.3 Ranking

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

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

Section 3.2 described how to identify Tuple Units and transformed them 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, and following the technique of indexing and ranking used in DISCOVER2 [13], SPARK [23], EASE [19] and by Liu and Yu in [22], 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 is defined in Equation 1. Where  $tf(k_i, d)$  represents the number of occurrences of the term  $k_i$  in a document  $d$ .

$$ntf(k_i) = 1 + \ln(1 + tf(k_i, d)) \quad (1)$$

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

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

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[22] and is usually set to 0.2.

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

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.

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

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  $K = k_1, k_2, \dots, k_n$ , 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.

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

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.

### 3.4 Indexing

To efficiently retrieve scores of the documents, we propose a *keyword-structure-aware-index* called *KSAI index*, which is similar to the traditional inverted index. The entries of KSAI are also the keywords that are contained in the virtual documents. KSAI 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  $d \in \{d_j | d_j \text{ contains the keyword } k_i\}$ . We indexing all the virtual documents using the Apache Hadoop Framework<sup>2</sup>. Hadoop is an open source framework for writing and running distributed applications that process large amounts of data using commodity hardware [18]. Hadoop uses an implementation of Map-Reduce programming model.

### 3.5 Query Processing

With the *KSAI* index, we can answer a keyword query as follows. Given a keyword query  $K = k_1, k_2, \dots, k_n$ , 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.

## 4. RESULTS

### 4.1 Experimental Study

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 [12],

<sup>2</sup><http://hadoop.apache.org/>

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

**Table 1: IMDB DataSet**

EASE [19] and RETUNE [20]. We used the Internet Movie DataBase (IMDB) to evaluate KESOSD. 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.

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 *KSAI* index is 741 seconds approximately and the size of the index is close to 355 MB. The *KSAI* 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.

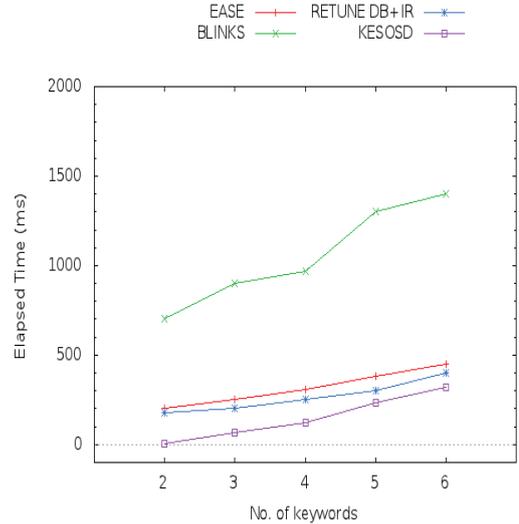
All the algorithms were coded in JAVA. All the experiments were conducted on a computer with an Intel(R) Core (TM) 2@2.33 GHz, CPU, 4 GM RAM running Ubuntu 11.10.

## 4.2 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. KESOSD 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 using SQL, since we include that relationships in the virtual documents in the *KSAI* index. KESOSD is faster than EASE, this is because EASE spend some time extracting the Steiner Graphs and eliminating the non-Steiner nodes. Besides, with the increase in the number of keywords, the elapsed time of BLINKS increases considerably compared with the elapsed time of

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

**Table 2: Several sample queries employed in the experiments**



**Figure 5: Search efficiency using 2 to 6 keywords**

KESOSD varies a few ms and always reaches higher search efficiency. This difference is due to the use of *KSAI* index to identify the answers in our approach. KESOSD 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 KESOSD 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 KESOSD 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 KESOSD varies slightly and always achieves high efficiency on the search.

## 4.3 Search accuracy

This section evaluate the search accuracy using standard precision. Because the lack of benchmark for keyword search over relational databases in which we can check the accuracy of our results, we consider that a document is precise with two constraints: First, if the document includes all the keywords in the textual attributes. And second, if the keywords in the document appears in the attribute that is semantically

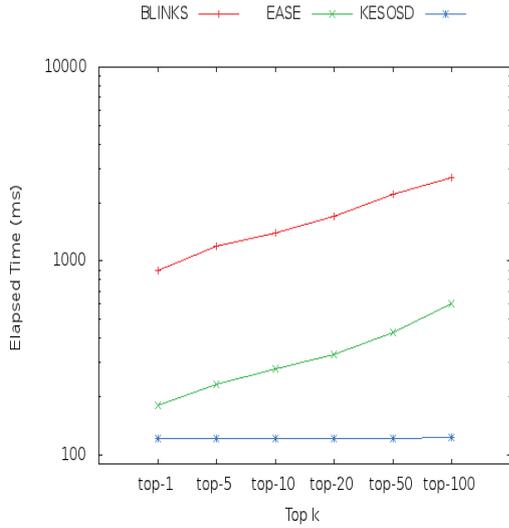


Figure 6: Top-k search efficiency using 4 keywords

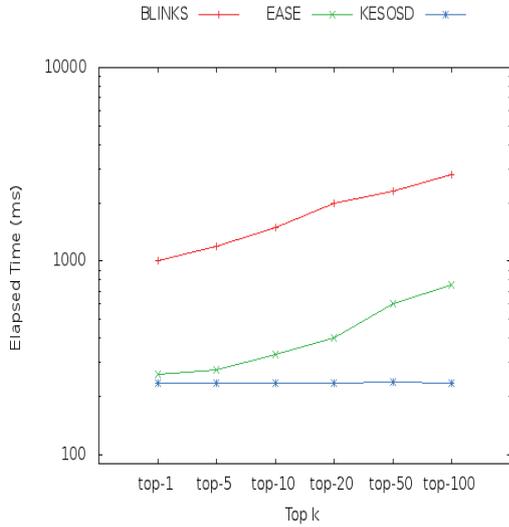


Figure 7: Top-k search efficiency using 5 keywords

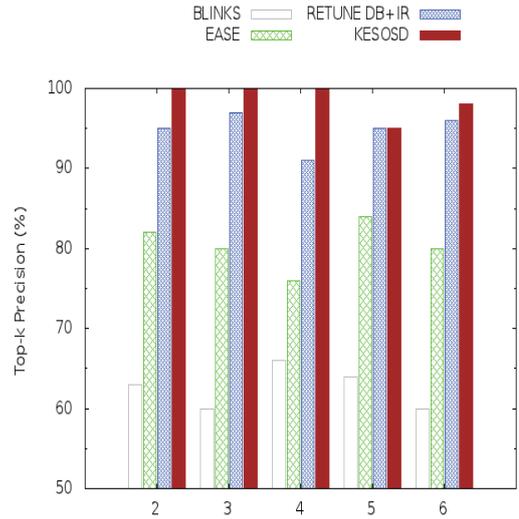


Figure 8: Top-k search precision

related to the keyword. This assumption is not automatic validated by the system. As example, if we have a keyword query with three keywords; two keywords in the movie name and one keyword in the occupation of the user that rates the movie respectively, we analyze the response and try to find the keywords in the corresponding attribute. If we can find the keywords we can assume that the document is precise. We also employed the metric top-k precision, which measures the ratio of the number of relevant answers among the first k answers with the highest scores of an algorithm to k. Figure 8 gives the experimental results. In this figure we can observe that our approach obtains higher search accuracy and outperforms the existing methods. KESOSD is better than BLINKS in various queries with different number of keywords. This is because our approach includes structural information in the index construction and the  $TF \cdot IDF$  based IR technique. KESOSD also outperforms EASE and RETUNE on terms of precision.

Figures 9 and 10 illustrate the experimental results, which show that KESOSD 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 KESOSD can always achieve high precision. This confirm the efficiency of our method.

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

## 5. 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

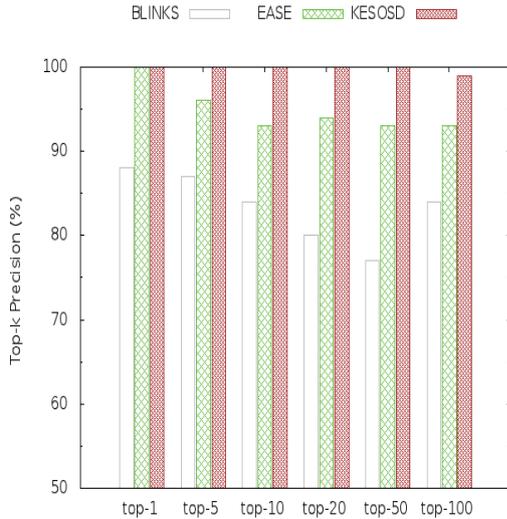


Figure 9: Top-k search precision using 4 keywords

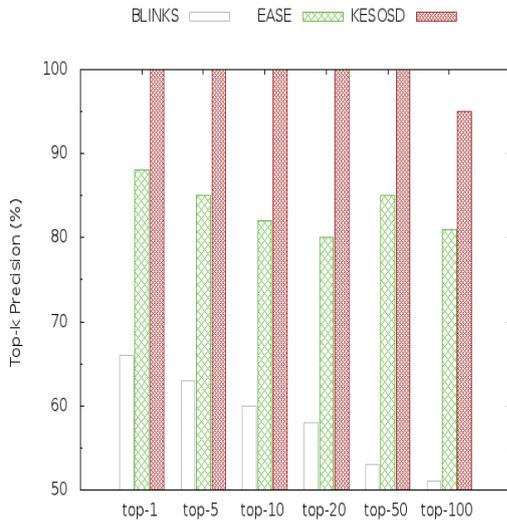


Figure 10: Top-k search precision using 5 keywords

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 [9]. The authors in [21, 12, 19, 8, 5] 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 then 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 [22], 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 [21]. As example the EASE approach proposed by [19] was reviewed and we find when the datagraph was generated from the IMDB Dataset<sup>3</sup>, it 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 very high computational effort and time consuming process, especially in the cases where a Minimal Steiner Graph contains hundred of nodes and arcs. BLINKS [12], BANKS [5], EASE [19] and CSTREE [21] 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 [28], this is because for a keyword query, the number of Candidate Networks can be very large. DISCOVER [14] and SPARK [23] are examples of this approach.

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 off-line [20].

<sup>3</sup><http://www.imdb.com/interfaces> (1,000,000 Data Set (475 MB))

The Tuple Units are proposed by Li et al. in [20]. 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 [26] and RETUNE [20] are examples of this approach. Following the RETUNE approach proposed by [19]. We tries 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 creates for a 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<sup>4</sup>. In this process, at least 4170 columns to store terms and 126,505 columns to store keyword pairs were required. This situation makes impossible to reproduce those experiments.

## 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 the KESOSD as an *Keyword Search Over Structured Data* to make an efficient processing of keyword queries. KESOSD is based on the classical Information Retrieval metrics and takes into account the structure extracted from structured information. KESOSD models structured data as graph and have proposed the use of an data index called *KSAI* 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 shows that KESOSD 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 KESOSD with other datasets including semi-structured and unstructured data. Also we have to continue working with some strategies to reduce of the size of the index as example with the using of Mutual Information.

## 7. ACKNOWLEDGMENTS

This research was partially supported by project number 187324 from Fondo Mixto Conacyt-Gobierno del Estado de Tamaulipas.

## 8. REFERENCES

- [1] S. Abiteboul and T. Allard. Webcontent: Efficient p2p warehousing of web data, 2008.
- [2] H. Achiezra and K. Golenberg. Exploratory keyword search on data graphs. In *Proceedings of the 2010 international conference on Management of data (SIGMOD)*, pages 1163–1166. ACM, 2010.
- [3] S. Agrawal and S. Chaudhuri. 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, 2002.
- [4] Z. Bao and J. Lu. Towards an effective xml keyword search. *IEEE Transactions on Knowledge and Data Engineering*, 22(8):1077–1092, 2010.
- [5] G. Bhalotia and A. Hulgeri. Keyword searching and browsing in databases using banks. In *Proceedings of the 18th International Conference on Data Engineering, ICDE '02*, pages 431–440, 2002.
- [6] S. Chaudhuri and R. Ramakrishnan. Integrating db and ir technologies: What is the sound of one hand clapping. In *Innovative Data Systems Research (CIDR)*, pages 1–12, 2005.
- [7] B. Ding and J. Xu. Finding top-k min-cost connected trees in databases, 2007.
- [8] X. Dong and A. Halevy. Indexing dataspace. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data, SIGMOD '07*, pages 43–54. ACM, 2007.
- [9] D. Du and X. Hu. *Steiner Tree problems in Computer Communication Networks*. World Scientific Publishing, 2008.
- [10] J. Feng and G. Li. Finding top-k answers in keyword search over relational databases using tuple units. *IEEE Transactions on Knowledge and Data Engineering*, 23:1781–1794, 2011.
- [11] M. Franklin and A. Halevy. From databases to dataspace: A new abstraction for information management. *SIGMOD Record*, 34:27–33, 2005.
- [12] H. He and H. Wang. 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, 2007.
- [13] V. Hristidis and L. Gravano. Efficient ir-style keyword search over relational databases. In *Proceedings of the 29th international conference on Very large data bases - Volume 29, VLDB '03*, pages 850–861. VLDB Endowment, 2003.
- [14] V. Hristidis and Y. Papakonstantinou. Discover: Keyword search in relational databases. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 670–681. VLDB Endowment, 2002.
- [15] V. Hristidis and Y. Papakonstantinou. Keyword proximity search on xml graphs. In *Proceedings. 19th International Conference Data Engineering*, pages 367–378, 2003.
- [16] V. Kacholia and S. Pandit. Bidirectional expansion for keyword search on graph databases, 2005.
- [17] B. Kimelfeld and Y. Sagiv. Efficiently enumerating results of keyword search over data graphs. *Information Systems*, 33:335–359, 2008.
- [18] C. Lam. *Hadoop in Action*. Manning Publications Co., 2011.
- [19] G. Li and J. Feng. 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, 2008.
- [20] G. Li and J. Feng. Retrieving and materializing tuple units for effective keyword search over relational databases. In *Lecture Notes in Computer Science*,

<sup>4</sup><http://dev.mysql.com/doc/refman/5.0/en/column-count-limit.html>

*Conceptual Modeling - ER*, pages 469–483, 2008.

- [21] G. Li and J. Feng. Providing built-in keyword search capabilities in rdbms, 2009.
- [22] F. Liu and C. Yu. 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, 2006.
- [23] L. Luo and X. Lin. 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, 2007.
- [24] K. S. M. Karnstedt. 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, 2008.
- [25] J. Park and S. goo Lee. Keyword search in relational databases. *Knowl. Inf. Syst.*, 26(2):175–193, 2011.
- [26] Q. Su and J. Widom. 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, 2005.
- [27] N. K. V. Hristidis. Keyword proximity search in xml trees. *IEEE Transactions on Knowledge and Data Engineering*, pages 525–539, 2006.
- [28] J. Xu and L. Qui. Keyword search in relational databases: A survey. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 33:67–78, 2010.
- [29] M. Zhong and M. Liu. 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, 2009.