

Searching Web Databases by Structuring Keyword-Based Queries

Pável Calado Altigran S. Silva Rodrigo C. Vieira
Alberto H. F. Laender Berthier A. Ribeiro-Neto

Department of Computer Science
Federal University of Minas Gerais
31270-901 Belo Horizonte MG Brazil
{rcvieira, pavel, alti, laender, berthier}@dcc.ufmg.br

ABSTRACT

On-line information services have become widespread in the Web nowadays. However, Web users are non-specialized have a great variety of interests. Thus, interfaces for Web databases must be simple and uniform. In this paper we present an approach, based on Bayesian networks, for querying Web databases using keywords only. According to this approach, the user inputs a query through a simple search-box interface. From the input query, one or more plausible structured queries are derived and submitted to Web databases. The results are then retrieved and presented to the user as ranked answers. Our approach reduces the complexity of existing on-line interfaces and offers a solution to the problem of querying several distinct Web databases with a single interface. The applicability of the proposed approach was demonstrated by experimental results with 3 databases, obtained with a prototype search system that implements it. We have found that from 77% to 95% of the time, one of the top three resulting structured queries is the proper one. Further, when the user selects one of these three top queries for processing, the ranked answers present average precision figures from 60% to about 100%.

Categories and Subject Descriptors

H.3.3 [Information Storage And Retrieval]: Information Search and Retrieval—*Query formulation*; H.3.5

[Information Storage And Retrieval]: Online Information Services—*Web-based services*

General Terms

Algorithms, Experimentation, Standardization

Keywords

Web databases, Query structuring, Structured queries

Permission to make digital or hard copies of all or part 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, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CIKM'02, November 4–9, 2002, McLean, Virginia, USA.
Copyright 2002 ACM 1-58113-492-4/02/0011 ...\$5.00.

1. INTRODUCTION

On-line information services, such as on-line stores and digital libraries, have become widespread in the Web nowadays. Such services allow a great number of users to access a large volume of data stored in local databases, also called *Web Databases*. Web users, however, are usually non-specialized and their interests vary greatly. Thus, two important problems are posed to designers of interfaces for Web databases: simplicity and uniformity. Interfaces for accessing Web databases are expected to be simple, since they are intended for layman users. In addition, if an on-line service is to provide access to different types of information (i.e., many distinct databases), its interface should be as uniform as possible. Otherwise, users will be required to learn how to use a different interface for each distinct database.

The most common solution for implementing on-line services that access Web databases is the use of customized forms, navigation menus, and similar browsing mechanisms. Although useful in some cases, this approach has some important shortcomings. Web sites that provide access to multiple databases, like *Amazon*¹, *MySimon*² and *Travelocity*³, include dozens of different forms, one for each type of product, which might be composed of a large number of fields. From the point of view of a Web user, this type of interface might seem rather complex. From the point of view of a Web developer, it increases the development time and maintenance costs.

Another common inconvenient of query interfaces for Web databases is the fact that the answer set is frequently too large. In a traditional database system, users have available the appropriate tools and query languages to restrict their search results. Thus, query interfaces for Web databases should also provide resources to help the user handle large answer sets. In Web search engines, document ranking [10, 13] is the most used method to deal with this problem. However, such method is not usually available in Web database interfaces.

In this work, we propose the use of keyword-based querying (as in a Web search engine) as an alternative to the use of interfaces based on multiple forms. Additionally, we pro-

¹<http://www.amazon.com>

²<http://www.mysimon.com>

³<http://www.travelocity.com>

pose to rank the possible large number of answers returned by a query according to a relevance criteria, as done in Web search engines. Our approach uses a Bayesian network (as in [10]) to model and derive structured queries from an unstructured query formed by the keywords specified by the user. The structured queries are then submitted to the Web database and the retrieved results are presented to the user as ranked answers. This means that the user needs only to fill a single search box to formulate a query. Our approach is thus able to provide on-line services with (1) an interface that is simple and intuitive to Web users and (2) the possibility of querying several heterogeneous databases using a single interface. As an additional advantage, such a simple interface is particularly suitable for portable devices, such as PDAs and cellular phones, where display space is limited.

To verify the feasibility of our approach, a simple prototype Web search system was implemented. Results obtained using this prototype on 3 different databases indicate that our approach allows accurately structuring user queries and return correct answers with a minimum intervention from the user. For a set of 83 unstructured queries all correct answers are retrieved with a tolerable interference of spurious results (30% of all answers are spurious). When the user interacts with the system to select the query to be processed (among the top 3 ranked alternatives), most correct answers are retrieved with minimal interference (less than 10% of the answers are spurious).

2. RELATED WORK

The use of keywords for query formulation is a rather common resource in information retrieval systems [10, 13], like Web search engines. In the context of structured databases, however, only more recently some proposals for the use of keyword-based queries have appeared [6, 1, 7]. The DataSpot system, described in [6], proposes the use of “plain language” queries and navigations to explore a *hyperbase* for publishing contents of a database on the Web. Agrawal et al. [1] introduce a system that allows querying databases through keywords. Their work, however, focus on relational databases and does not provide any ranking for the *approximate* answers. The work described in [7] proposes the extension of XML-QL, a well known query language for XML with keyword based searching capabilities. Our approach is distinct since it adopts a much simpler query language. Although this can make our approach less expressive, it also makes it more accessible to regular Web users. Most important, we propose to rank the answers to avoid the effect of spurious data, which will necessarily be present.

The vector space model has been the most widely used solution to the problem of ranking query results in text databases [2]. This model is also employed in the work described in [5] to determine the similarity between objects in a database composed of relations whose attribute values are free text. For ranking structured objects returned as answers of queries over a local database, the work described in [4] combines application-oriented scoring functions. An extension of such an idea is proposed in [3] for the case of Web accessible databases. In this case the scoring functions may be based on evidences coming from distinct Web databases. In [8], object ranking is based on a measure of the proximity of the objects, that are seen as nodes, in the database, that is seen as a graph. This approach led to a search system similar to ours in functionality, since it also

allows keyword-based queries, but that is based on a rather distinct approach.

In our work, the ranking of query results is performed using vectorial techniques, but within a Bayesian network approach [10]. Bayesian networks have been first used to model information retrieval problems by Turtle and Croft [14] and, as demonstrated in [11], they can be used to represent any of the classic models in information retrieval. Bayesian network models are especially useful when we need to determine the relevance of the answers in view of several independent evidences [13, 10]. In our case, Bayesian networks are used to compute the likeliness of a structured query representing the user’s needs and the relevance of the returned answers.

3. OVERVIEW OF THE APPROACH

We now present an overview of our approach. To simplify the discussion, we start with some basic definitions.

3.1 Definitions

Consider a database D accessible through a Web query interface (for instance, an HTML form). We define this database as a collection of *objects*

$$D = \{o_1, o_2, \dots, o_n\}, \quad n \geq 1$$

Each object o_i is a set of *attribute-value* pairs

$$o_i = \{\langle A_1, v_{1i} \rangle, \dots, \langle A_{k_i}, v_{k_i i} \rangle\}, \quad k_i \geq 1$$

where each A_j is an attribute and each v_{ji} is a value belonging to the domain of A_j . We note the attributes do not need, necessarily, to be the same for all objects.

For some attributes, instead of a single value, we may have a list of values. For instance, in an object representing a movie, the attribute *actor* might be a list of names. To represent this using our notation, we allow a same attribute to appear several times, here called a *value list*. Thus, if attribute A_j , in object o_i , has n different values, we can represent object o_i as:

$$o = \{\dots, \langle A_j, v_1 \rangle, \langle A_j, v_2 \rangle, \dots, \langle A_j, v_n \rangle, \dots\}$$

We define a *database schema* as the set of all attributes that compose any of the stored objects. Thus, the schema of a database D is defined as

$$S_D = \{A \mid A \text{ is an attribute of some object } o \in D\}.$$

We define an *unstructured query* U as a set of keywords (or *terms*)

$$U = \{t_1, t_2, \dots, t_k\}.$$

As for an object, a *structured query* Q is defined as a set of ordered pairs:

$$Q = \{\langle A_1, v_{1q} \rangle, \dots, \langle A_m, v_{mq} \rangle\}, \quad m \geq 1,$$

where each A_j is an attribute and each v_{jq} a value belonging to the domain of A_j .

This simplified definition of a database allows us to ignore the details of how its structure is represented. Also, we can regard the database simply as data repository available through some high level user interface. This Web database can be composed of a relational table, a set of relational tables, an XML repository, etc.

Throughout the text, we informally use the term *application domain* of a database to refer to the main topic associated with the objects in the database. For instance, a

database with the application domain Book stores objects with information on books. The database to be queried by the user is called the *target database*. As an example, consider a database D with the application domain Book. An object o in D could be $o = \{\langle \text{Title}, \text{"I, Robot"} \rangle, \langle \text{Author}, \text{"Isaac Asimov"} \rangle\}$. An example of an unstructured query is $K = \{\text{"asimov"}, \text{"robot"}\}$. An example of a structured query is $Q = \{\langle \text{Author}, \text{"Isaac Asimov"} \rangle\}$.

3.2 Querying the Target Database

Our proposed approach consists of four steps: (1) inputting the unstructured user query, (2) building a set of possible structured queries derived from it, (3) selecting one, or possibly more, of the candidate structured queries as being the "best" ones, and (4) processing the results of these selected queries. We now discuss each one of these steps.

Step 1 consists simply of receiving the unstructured query from the user as a set of keywords. This can be easily accomplished using a simple search box interface, in which queries are specified entering the keywords.

In step 2, for a given unstructured query $K = \{t_1, t_2, \dots, t_n\}$, we need to determine a set P_K of possible structured queries. Let D be a database, with a schema S_D . A simple way to determine P_K is to build all possible combinations of all query terms $t_i \in K$ and all attributes $A_j \in S_D$. Clearly, in the worst case, this operation would be of exponential complexity. In practice, however, it is possible to discard many combinations of terms and attributes (i.e., many attribute value pairs) in advance. For instance, if there are little or no occurrences of the word "Hamlet" in the Author field, we can discard all possible queries that make such an assignment.

Once we have determined a set $P_K = \{Q_1, \dots, Q_n\}$, we proceed to step 3 that consists of selecting which queries are most likely to match the user needs. For instance, the unstructured query $K = \text{"asimov robot"}$, might have as the most probable structured query $Q_1 = \{\langle \text{Author}, \text{"asimov"} \rangle, \langle \text{Title}, \text{"robot"} \rangle\}$. More precisely, the query structuring process consists in determining the database attributes which are most likely to correspond to each term in set K . Once the "best" attribute-value pairs have been determined, we build a set of candidate structured queries by combining these pre-selected attribute-value pairs. The best candidate structured query maximizes the likelihood that its component attribute-value pairs correspond to the user's query intention.

In step 4, we take the ranking of structured queries in P_K and submit the top ranked ones. We can, for instance, pick the best ranked query and submit it, without user interference. Alternatively, we can present the top ranked queries to the user and let him chose one of them for processing. In both cases, a structured query Q is submitted to the target database D and a set of returned objects RD_Q is obtained. The objects in RD_Q are also ranked according to a probability that they satisfy the user's needs as we later discuss.

Ranking the returned objects is specially important when the query Q can be sent to more then one target database, and the results merged together. For instance, in an application domain like Computer science bibliographic references, the same unstructured query can be used to search the ACM and IEEE digital libraries. Notice that this is accomplished without the need to define a unifying common database schema.

To rank the candidate queries in step 3 (from set P_k) and

the returned objects in step 4 (from set RD_Q), we adopt the framework of Bayesian belief networks [9]. Belief networks provide a flexible and formally sound framework, which can be conveniently adapted to combine distinct sources of evidence as demonstrated in [10, 13, 14]. In Section 4, we discuss the Bayesian model proposed for ranking candidate queries. In Section 5, we present the Bayesian model for ranking the retrieved objects.

3.3 The Prototype Web Search System

Based on the approach describe above, a prototype search system was implemented. Its architecture is shown in Figure 1.

In this system, queries are specified trough a single search box. Unstructured queries entered by the user are passed to the Query Structuring module. This module builds the set of possible queries and ranks them. A set of ranked structured queries is then passed to the query processing module, which displays to the user the top ranked structured queries. The user selects a query of his preference by clicking on it. The selected query is then submitted to the target database. The returned results are ranked and presented to the user.

4. RANKING STRUCTURED QUERIES

The query structuring process associates structure to a given query K , consisting only of keywords. Several structured queries are generated for a same unstructured query. For this reason, we need a consistent way of ranking the queries according to their similarity to the database, in other words, their similarity to the application domain in question. The ranking of structured queries is accomplished through the use of the Bayesian network model, shown in Figure 2. Although the network can be easily expanded to model any database schema, for simplicity, here we show only two attributes, A_1 and A_2 .

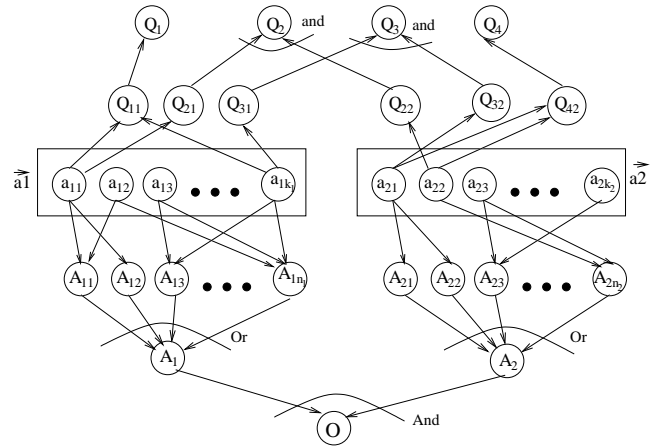


Figure 2: Bayesian network model for query structuring.

The network consists of a set of nodes, each representing a piece of information from the problem to be solved. To each node in the network is associated a binary random variable. This variable takes the value 1 to indicate that the corresponding information will be accounted for in the ranking computation. In this case, we say that the information was *observed*. To simplify the notation, we define T_i as the set

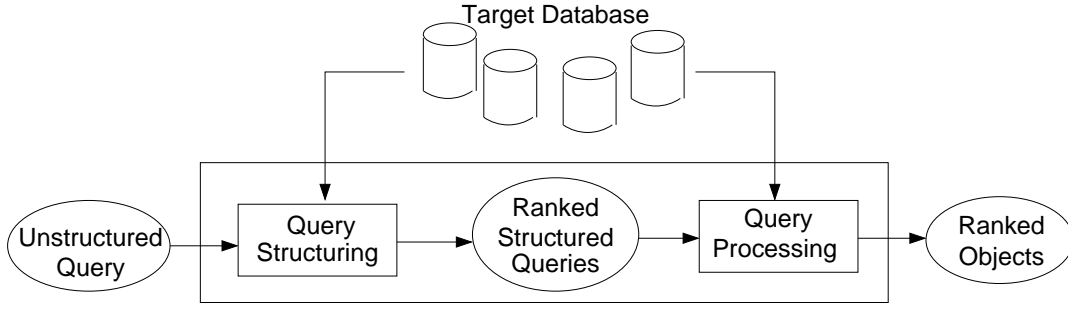


Figure 1: Search system architecture.

of all terms in the database that compose the values in the domain of attribute A_i . In this case, each value is considered as a string and the terms are the words in the string.

In the network of Figure 2, the database is represented by node O . Each node A_i represents an attribute in the database. Each node a_{ij} represents a term in T_i . Each node Q_i represents a structured query to be ranked. Each node Q_{ij} represents the portion of the structured query Q_i that corresponds to the attribute A_j . Vectors \vec{a}_1 and \vec{a}_2 represent a possible state of the variables associated to the nodes a_{1i} and a_{2i} , respectively.

We can think of this network as modeling, for instance, a database on books, with attributes $A_1 = \text{Title}$ and $A_2 = \text{Author}$. In this case, node A_{11} represents the title of a stored book, like “I, Robot”, where the term “I” is represented by node a_{11} and the term “Robot” is represented by node a_{12} . In a similar fashion, node Q_2 represents the structured query $Q_2 = \{\langle \text{Title}, \text{“I, Robot”} \rangle, \langle \text{Author}, \text{“Asimov”} \rangle\}$, where Q_{21} is the part referring to the Title attribute, Q_{22} the part referring to the Author attribute. Node a_{22} is the term “Asimov”.

The similarity of a structured query Q_i with the database O can be seen as the probability of observing Q_i , given that the database O was observed, $P(Q_i|O)$. Examining the network in Figure 2, we can derive the equation:

$$\begin{aligned} P(Q_i|O) &= \alpha \times \sum_{\vec{a}_1, \vec{a}_2} P(Q_i|\vec{a}_1, \vec{a}_2) \times P(O|\vec{a}_1, \vec{a}_2) \times P(\vec{a}_1, \vec{a}_2) \\ &= \alpha \times \sum_{\vec{a}_1, \vec{a}_2} P(Q_{i1}|\vec{a}_1) \times P(Q_{i2}|\vec{a}_2) \times \\ &\quad P(A_1|\vec{a}_1) \times P(A_2|\vec{a}_2) \times P(\vec{a}_1) \times P(\vec{a}_2) \end{aligned} \quad (1)$$

where α is a normalizing constant (see [9, 10] for details).

Eq. (1) is the general equation for ranking a structured query. The conditional probabilities can now be defined, according to the values stored in the database. We start with the probability of observing the part of query Q_i assigned to an attribute A_j , given that a set of terms, indicated by \vec{a}_j was observed:

$$P(Q_{ij}|\vec{a}_j) = \begin{cases} 1 & \text{if } \forall k, g_k(\vec{a}_j) = 1 \text{ iff } t_{jk} \text{ occurs in } Q_{ij} \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

where $g_k(\vec{a}_j)$ gives the value of the k -th variable of the vector \vec{a}_j and t_{jk} is the k -th term in T_j . This equation guarantees that just the states where the only active terms are those of the query Q_i are taken into consideration.

The probability of observing the attribute A_i , given the

terms indicated by \vec{a}_i , is defined as a disjunction of all probabilities for each possible value of A_i , i.e.:

$$P(A_i|\vec{a}_i) = 1 - \prod_{1 \leq j \leq n_i} (1 - P(A_{ij}|\vec{a}_i)) \quad (3)$$

where n_i is the number of values in the database for attribute A_i . If we consider that $P(A_{ij}|\vec{a}_i)$ measures the similarity between the value A_{ij} and the terms indicated by \vec{a}_i , then Eq. (3) tells us that, if the terms in \vec{a}_i fit exactly the value A_{ij} , the final probability is 1. If not, the final probability will accumulate all the partial similarities between the terms in \vec{a}_i and the value A_{ij} .

To define the probability of observing a value A_{ij} , given a set of terms indicated by \vec{a}_i , $P(A_{ij}|\vec{a}_i)$, we use the cosine measure, as defined for the vector space model in information retrieval systems [12]. The value A_{ij} for attribute A_i is seen as a vector of $|T_i|$ terms. To each term t_k in A_{ij} we assign a weight w_{ik} that reflects the importance of the term for attribute A_i , in the database, i.e.:

$$w_{ik} = \begin{cases} \frac{\log(1+f_{ki})}{\log(1+n_i)} & \text{if } t_k \text{ occurs in } A_{ij} \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

where f_{ki} is the number of occurrences of term t_k in the values of the attribute A_i , and n_i is the total number of values for attribute A_i , in the database. Notice that we do not consider the traditional *idf* normalization [12], since this would penalize the most frequent terms, which are the most important for our approach.

The probability of observing A_{ij} is, therefore, defined as the cosine of the angle between vector A_{ij} and vector \vec{a}_i , i.e.:

$$P(A_{ij}|\vec{a}_i) = \cos(A_{ij}, \vec{a}_i) = \frac{\sum_{t_k \in T_i} w_{ik} g_k(\vec{a}_i)}{\sqrt{\sum_{t_k \in T_i} w_{ik}^2}} \quad (5)$$

Finally, since we have no *a priori* preference for any set of terms, Eq. (6) defines the probability of vector \vec{a}_i as a constant:

$$P(\vec{a}_i) = \frac{1}{2^{|T_i|}} \quad (6)$$

In summary, the probability $P(Q_i|O)$ is the conjunction of the largest similarities between the attribute values in the database and the values assigned to the respective attributes in the structured query. This is translated by the equation:

$$\begin{aligned} P(Q_i|O) &= \eta \times [1 - \prod_{1 \leq j \leq n_1} (1 - \cos(A_{1j}, \vec{a}_1))] \times \\ &\quad [1 - \prod_{1 \leq j \leq n_2} (1 - \cos(A_{2j}, \vec{a}_2))] \end{aligned} \quad (7)$$

where \vec{a}_1 and \vec{a}_2 are the states where only the query terms referring to attributes A_1 and A_2 , respectively, are active, n_1 and n_2 are the total number of values for attributes A_1 and A_2 in the database, and η accounts for the constants α and $P(\vec{a}_j)$.

We can now rank all the structured queries by computing $P(Q_i|O)$ for each of them. The user can then select one query for processing among the top ranked ones, or the system can simply process the first query.

5. RANKING RETURNED OBJECTS

Once a structured query is selected, it is submitted to the target database. The set of objects returned can then be ranked according to the probability of satisfying the user needs. To this effect, the structured query and the returned objects are modeled using the Bayesian network model shown in Figure 3. Although the network can be easily expanded to model any database schema, for simplicity, here we show only two attributes, A_1 and A_2 . To simplify the notation, we define T_i as the set of all terms, in the returned objects, that compose the values in the domain of the attribute A_i .

When ranking the returned objects, special attention needs to be paid to value lists, i.e., attributes that are constituted by a list of values, as explained in Section 3.1. To exemplify how this type of attribute is treated, in the network we represent attribute A_2 as a value list attribute.

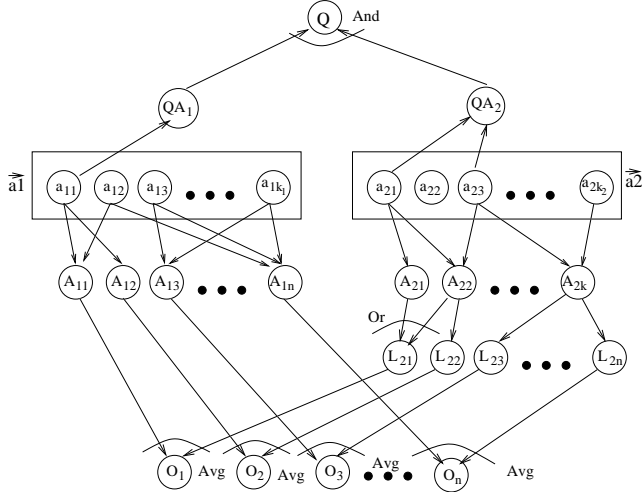


Figure 3: Bayesian network model for result ranking.

In the network in Figure 3, node Q represents the structured query, submitted for processing. Each node QA_i represents the part of the structured query Q relative to the attribute A_i . Each node a_{ij} represents a term in T_i . Each node A_{ij} represents the value of the attribute A_i in object O_j . Since attribute A_2 is a list attribute, we add one more level to the network. Each node L_{2j} represents the list of values assigned to the attribute A_2 in object O_j . Finally, each node O_j represents a returned object. As before, with each node in the network is associated a random variable that takes the value 1 to indicate that information regarding the respective node was observed. Vectors \vec{a}_1 and \vec{a}_2 represent a possible state of the variables associated with nodes a_{1i} and a_{2i} , respectively.

An object O_j is ranked according to the probability of satisfying the structured query Q , i.e., the probability of observing O_j , given that Q was observed, $P(O_j|Q)$. Examining the network in Figure 3, we have that:

$$\begin{aligned} P(O_j|Q) &= \beta \times \sum_{\vec{a}_1, \vec{a}_2} P(Q|\vec{a}_1, \vec{a}_2) \times P(O_j|\vec{a}_1, \vec{a}_2) \times P(\vec{a}_1, \vec{a}_2) \\ &= \beta \times \sum_{\vec{a}_1, \vec{a}_2} P(QA_1|\vec{a}_1) \times P(QA_2|\vec{a}_2) \times \\ &\quad \frac{P(A_{1j}|\vec{a}_1) + P(L_{2j}|\vec{a}_2)}{2} \times P(\vec{a}_1) \times P(\vec{a}_2) \end{aligned} \quad (8)$$

where β is a normalizing constant. The average between $P(A_{1j}|\vec{a}_1)$ and $P(L_{2j}|\vec{a}_2)$ allows us to accumulate the evidences associated with each attribute value for the final probability.

We define a list as a disjunction of its values. Therefore, the probability for the list attribute is given by:

$$P(L_{2j}|\vec{a}_2) = 1 - \prod_{\forall k \in \mathcal{V}} (1 - P(A_{2k}|\vec{a}_2)) \quad (9)$$

where \mathcal{V} is the set of indexes for the values in the list represented by L_{2j} . Eq. (9) determines that, for a value list to be observed, it is enough that one of its values is observed.

Eqs. (8) and (9) are the general formula to rank a returned object. The conditional probabilities can now be defined. We start by the probability of observing the part of Q relative to an attribute A_i , given that a set of terms, indicated by \vec{a}_i , was observed:

$$P(QA_i|\vec{a}_i) = \begin{cases} 1 & \text{if } \forall k, g_k(\vec{a}_i) = 1 \text{ iff } t_{ik} \text{ occurs in } QA_i \\ 0 & \text{otherwise} \end{cases} \quad (10)$$

where $g_k(\vec{a}_i)$ indicates the value of the k -th variable of the vector \vec{a}_i , and t_{ik} is the k -th term in T_i . This equation guarantees that only the states where the only active terms are those of the query Q will be taken into consideration.

As for query structuring, the probability observing the value A_{ij} given that the terms indicated by \vec{a}_i were observed, $P(A_{ij}|\vec{a}_i)$, is defined using the cosine measure. The value A_{ij} of the object O_j (value associated with the attribute A_i) is seen as a vector of $|T_i|$ terms. This time, however, we are interested in determining whether each term t_k occurs, or not, in the value A_{ij} . Therefore, we define the term weight w_{ik} simply as:

$$w_{ik} = \begin{cases} 1 & \text{if } t_k \text{ occurs in } A_{ij} \\ 0 & \text{otherwise} \end{cases} \quad (11)$$

The probability of A_{ij} is defined as:

$$P(A_{ij}|\vec{a}_i) = \cos(A_{ij}, \vec{a}_i) = \frac{\sum_{t_k \in T_i} w_{ik} g_k(\vec{a}_i)}{\sqrt{\sum_{t_k \in T_i} w_{ik}^2}} \quad (12)$$

As in Section 4, since we have no preference for any particular set of terms, Eq. (13) defines the probability associated with the vector \vec{a}_i as a constant:

$$P(\vec{a}_i) = \frac{1}{2^{|T_i|}} \quad (13)$$

In conclusion, the probability $P(O_j|Q)$ is the average of the similarities between the attribute values in object O_j

and the values for the respective attributes in the structured query, i.e.:

$$P(O_j|Q) = \eta \times \frac{1}{2} \times \left[\cos(A_{1j}, \vec{a}_1) + \left[1 - \prod_{v_k \in \mathcal{V}} (1 - \cos(A_{2k}, \vec{a}_2)) \right] \right] \quad (14)$$

where \vec{a}_i is the state where the only active terms are those in the query part referring to the attribute A_i , η summarizes the constants β and $P(\vec{a}_i)$, and \mathcal{V} is the set indexes for the values in the list for attribute A_2 .

Once we compute the values of $P(O_j|Q)$ for all the returned objects, they can be presented to the user as a ranked list.

6. EXPERIMENTAL RESULTS

To evaluate our rankings of structured queries and corresponding answers, we used the implemented prototype system to run experiments. We first describe the experimental setup and then discuss the results of these experiments.

6.1 Experimental Setup

Experiments were performed using three distinct databases with application domains Books, CDs, and Movies. 20 structured test queries were created for Books and CDs database, and 43 for the Movies database. To each structured query Q_i (for database D with schema S_D), we assigned an unstructured query U_i , defined as

$$U_i = \{t \mid t \text{ is a term in } v_j \wedge \langle A_j, v_j \rangle \in Q_i\}$$

where $A_j \in S_D$. The query U_i is the set of all terms that compose the attribute values of Q_i . For instance, to the structured query $Q = \{ \langle \text{Title}, \text{"I, Robot"} \rangle, \langle \text{Author}, \text{"Isaac Asimov"} \rangle \}$, corresponds the unstructured query $U = \{ \text{"I"}, \text{"Robot"}, \text{"Isaac"}, \text{"Asimov"} \}$. In our experiment, the query Q_i was taken as the correct structured formulation of the unstructured query U_i .

To determine the set of correct answers for U_i , the corresponding structured query Q_i was submitted to the target database. The set RU_i of objects returned, here called set of *relevant objects*, was taken as the set of correct answers for the query U_i . This set is used to evaluate how many relevant objects (i.e., objects in set of correct answers) the system is able to retrieve when only the unstructured query U_i is processed.

The most commonly used measures to determine the quality of a ranked set of objects are precision and recall figures [2]. These measures are defined as follows. For a given query U_i , let RU_i be the set of correct objects and let SA_i be the total set of objects returned by the system (i.e., the system answer). Precision is defined as:

$$p = \frac{|SA_i \cap RU_i|}{|SA_i|} \quad (15)$$

which is the fraction of the set of objects retrieved that is correct. Recall is defined as:

$$r = \frac{|SA_i \cap RU_i|}{|RU_i|} \quad (16)$$

which is the fraction of correct objects that was retrieved. Usually, we are interested in the values of precision taken at 11 standard recall points, 0%, 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, and 100%. This precision/recall curve

gives us the quality of the ranked set of answers, as we descend along the ranking. Precision at 0% is defined as the precision when we find the first relevant object in the ranked answer set. In our experiments, the precision and recall values can be seen as correlation measures between the results of the manually structured queries and the queries structured by our algorithm.

Table 1 details the characteristics of the test databases and queries. The queries were chosen to have a small number of words since Web users tend to use short queries. We note that the high number of relevant objects per query for the movie database is due to the size of the database and the use of some generic queries, like $Q = \{ \langle \text{Genre}, \text{"Horror"} \rangle \}$.

6.2 Results

The tests were performed by submitting the unstructured queries to the system and evaluating: (1) the system's capability of correctly structuring the queries, and (2) the quality (precision) of the ranked answers.

When an unstructured query U_i is submitted, a set of ranked structured queries SQ_i is returned. To evaluate point (1), the system's capability of correctly structuring the user queries, we measure the number of attributes correctly assigned to the queries in SQ_i . Table 2 shows the percentage of correct queries and correctly assigned attribute-value pairs for the top ranked positions in SQ_i , for the three test databases, Movies (M), CDs (C) and Books (B).

We see that 60.5%, 65%, and 70% of the highest ranked queries were correctly structured, i.e., were equal to the original structured query, for the Movies, CDs and Books domains, respectively. If we consider the top three positions in the ranking, the percentage of correctly structured queries is always above 76%, and as high as 95% for the Books domain. For the queries ranked in the first place the system was able to correctly determine the correct value of, at least, 74% of the attributes. Considering the first three queries in the ranking, 84.1%, 92.5%, and 96.7% of the values were correctly assigned for the three domains.

These results indicate that the system is capable, most of the time, of ranking in first place the correct query. Also, by simply showing the user the first three queries in the ranking, in 76.7% to 95% of the time the user will be able to select the correct one. Even for the small number of cases where not all the values were correctly assigned, due to our result ranking approach, some relevant objects might still be returned, as we discuss below. Experiments with the Movies database have shown poorer results mainly due to then intersection between the attribute domains. For instance, the fact that many values in attribute Writer are also values for attribute Director make harder the task of distinguishing between them.

To evaluate the result generated by the structured queries assembled by the system, we computed a disjunction of the attributes in each query, i.e., any object containing at least one of the values in the query was returned. The answer set contains, therefore, all the relevant, together with many non-relevant objects. This allows us to evaluate the effectiveness of our result ranking algorithm, which should be able to assign a highest score to the relevant objects.

The graphics in Figures 4, 5, and 6 show the average precision for the test queries in the Movies, CDs and Books database, respectively. The curve labeled "First" shows the precision for the results obtained when the first ranked query

Domain	Number of objects	Attributes per object	Number of queries	Keywords per query	Relevants per query
Book	107,206	3	20	2.35	146.75
CDs	116,983	3	20	2.25	43.55
Movies	116,119	6	43	2.37	10,106.23

Table 1: Characteristics of the databases used in the experiments.

Rank Position	1st			1st+2nd			1st + 2nd + 3rd		
	M	C	B	M	C	B	M	C	B
Correct queries (%)	60.5	65.0	70.0	67.4	80.0	85.0	76.7	85.0	95.0
Correct attributes (%)	74.0	81.0	87.0	76.2	89.8	95.5	84.1	92.5	96.7

Table 2: Percentage of correctly assigned queries and attributes.

in SQ_i is submitted. We see that, for all cases, 100% of all correct answers (i.e., 100% recall) are retrieved. For the Movies domain, although only 60% of the queries were correct, about 40% of the objects on the top of the ranking are relevant, when 50% of all the relevant objects were retrieved. For the CDs and Books domain precision is always above 70% when 50% and 70% of the relevant objects were retrieved, respectively.

The curve labeled “User” shows the results when the user selects the query to be processed among the three highest ranked queries in SQ_i . Results are much improved. For the Movies domain, the system is now able to show a precision of about 60% when all the relevant objects were retrieved. For the CDs domain, precision is almost 100% throughout all the ranked answer set, and for the Books domain the system is able to show the user 90% relevant objects, when 70% of all the relevant objects were retrieved.

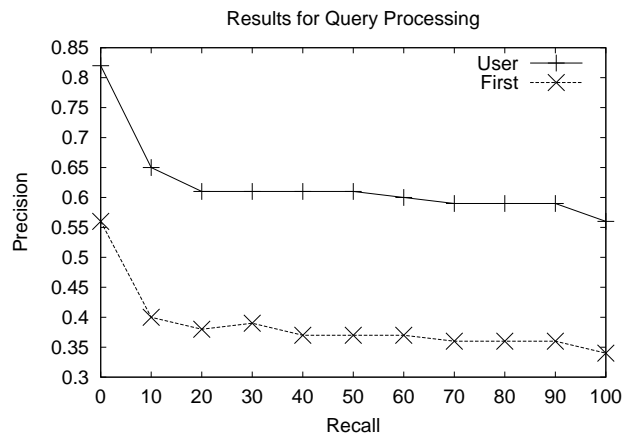


Figure 4: Precision/recall values for the returned results for Movies.

These values indicate that, not only our result ranking strategy allows the system to show the user many relevant objects, even if not always the correct query is chosen, but also that, with a very small user interaction, the quality of the results can be highly similar to the results that would be obtained by allowing the user to build a structured query, a much more laborious task. Although the results were poorer for the Movies domain, some preliminary experiments indi-

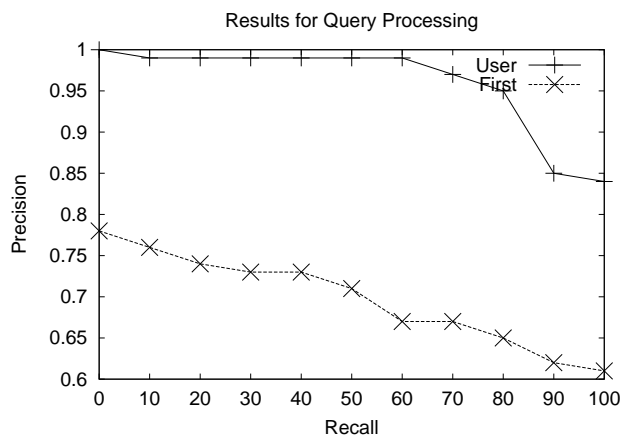


Figure 5: Precision/recall values for the returned results for CDs.

cate that these can be highly improved by using information from a database query log. These results, however are left as future work.

7. CONCLUSIONS AND FUTURE WORK

In this work, we have presented a novel approach to allow the querying of Web databases using unstructured queries, i.e., queries constituted only of keywords. The approach is based on a Bayesian network model, which combines evidence taken from object attributes to provide structure for the keyword-based user query. Once the structured query is determined, it is submitted to one or more databases. Since the returned list of results can be quite extensive, we also propose a Bayesian network model to rank the results according to their probability of satisfying the user’s needs.

Our query structuring and result ranking models provide a framework for querying Web databases that: (1) allows the formulation of queries using a very simple interface (one single text search box), (2) allows the use of a single interface to query any number of different databases, and (3) provides the results as a ranked set, where the objects most likely to answer the user’s query are shown first. These qualities make our proposal ideal for querying Web databases such as the ones available from on-line stores or digital libraries.

To test our approach, we implemented a prototype Web

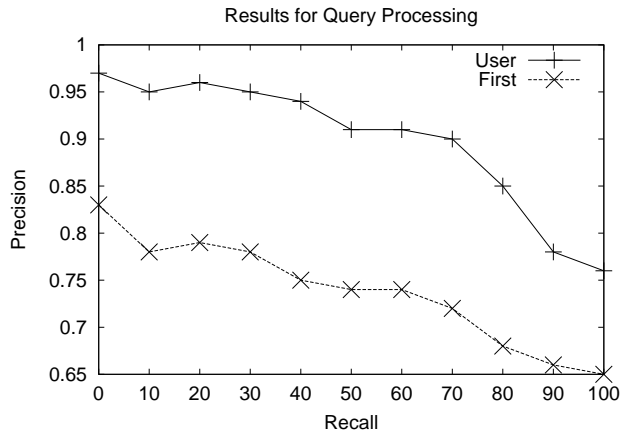


Figure 6: Precision/recall values for the returned results for Books.

search system, capable of querying several different databases with a single interface. Experimental results using 3 different databases show that, for each unstructured query submitted, our system was able to show the user the correct query among the first three proposed, from 77% to 95% of the time. Further, when the user selects one of these three top queries for processing, the ranked answers present average precision figures from 60% to about 100%.

As future work, we are currently considering three problems. First, in most on-line services, the target database might be available only through its Web interface. To determine the probabilities required for our approach we need, therefore, a local database that contains objects representative of the application domain in question. This local database can be, for instance, a subset of the target database and does not need to be rigidly structured, but can be simply a list of possible values for each attribute. In fact, some preliminary studies in this direction have already shown that, not only can we achieve very good results, more efficiently, using only a representative subset of values, but also that the results can be improved if this local database is built based on a query log.

Second, in the discussion so far, we have mentioned query structuring and processing applied to a single application domain. One important advantage of our approach is that it is also useful for querying several databases that cover multiple application domains. E-commerce sites, for instance, can have several types of products for sale (CD, books, DVD, etc.) and our approach allows using a same single interface to search all these products in a uniform way. For multiple application domains, a set of possible structured queries can be built for each domain and then ranked. By ranking the queries, our approach is also able to accurately determine the correct domain, since the highest ranked queries will most likely be those in the domain intended by the user, as also indicated by some preliminary experiments.

Third, additions to the query language might increase its expressiveness, without greatly increasing its complexity. For instance, the introduction of numeric operators, or allowing the user to restrict certain terms to certain attributes. This extension will, of course, imply some changes on the proposed Bayesian network models.

8. REFERENCES

- [1] AGRAWAL, S., CHAUDHURI, S., AND DAS, G. DBXplorer: A System For Keyword-Based Search Over Relational Databases. In *18th International Conference on Data Engineering* (San Jose, California, 2002).
- [2] BAEZA-YATES, R., AND RIBEIRO-NETO, B. *Modern Information Retrieval*. Addison Wesley, New York, NY, 1999.
- [3] BRUNO, N., GRAVANO, L., AND MARIAN, A. Evaluating top-k queries over web-accessible databases. In *20th International Conference on Data Engineering* (San Jose, California, USA, 2002), p. To appear.
- [4] CHAUDHURI, S., AND GRAVANO, L. Evaluating top-k selection queries. In *Proceedings of 25th International Conference on Very Large Data Bases* (Edinburgh, Scotland, UK, 1999), pp. 397–410.
- [5] COHEN, W. W. Reasoning about Textual Similarity in a Web-Based Information Access System. *Autonomous Agents and Multi-Agent Systems* 2, 1 (1999), 65–86.
- [6] DAR, S., ENTIN, G., GEVA, S., AND PALMON, E. Dtl's dataspot: Database exploration using plain language. In *Proceedings of 24th International Conference on Very Large Data Bases* (New York, New York, USA, 1998), pp. 645–649.
- [7] FLORESCU, D., KOSSMANN, D., AND MANOLESCU, I. Integrating Keyword Search into XML Query Processing. *WWW9 / Computer Networks* 33, 1-6 (2000), 119–135.
- [8] GOLDMAN, R., SHIVAKUMAR, N., VENKATASUBRAMANIAN, S., AND GARCIA-MOLINA, H. Proximity search in databases. In *Proceedings of 24th International Conference on Very Large Data Bases* (New York, New York, USA, 1998), pp. 26–37.
- [9] PEARL, J. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, 1988.
- [10] RIBEIRO-NETO, B., AND MUNTZ, R. A belief network model for IR. In *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (Zurich, Switzerland, August 1996), pp. 253–260.
- [11] RIBEIRO-NETO, B., SILVA, I., AND MUNTZ, R. *Soft Computing in Information Retrieval: Techniques and Applications*, 1st ed. Springer Verlag, 2000, ch. 11—Bayesian Network Models for IR, pp. 259–291.
- [12] SALTON, G., AND MCGILL, M. J. *Introduction to Modern Information Retrieval*, 1st ed. McGraw-Hill, 1983.
- [13] SILVA, I., RIBEIRO-NETO, B., CALADO, P., MOURA, E., AND ZIVIANI, N. Link-based and Content-Based Evidential Information in a Belief Network Model. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (Athens, Greece, July 2000), pp. 96–103.
- [14] TURTLE, H., AND CROFT, W. B. Evaluation of an Inference Network-Based Retrieval Model. *ACM Transactions on Information Systems* 9, 3 (July 1991), 187–222.