Answering Structured Queries on Unstructured Data

Jing Liu and Xin Dong
University of Washington
Seattle, WA 98195
{liujing, lunadong}@cs.washington.edu

Alon Halevy
Google Inc.
Mountain View, CA 94022
halevy@google.com

June 7, 2006

Abstract

There is growing number of applications that require access to both structured and unstructured data. Such collections of data have been referred to as *dataspaces*, and *Dataspace Support Platforms (DSSPs)* were proposed to offer several services over dataspaces, including search and query, source discovery and categorization, indexing and some forms of recovery. One of the key services of a DSSP is to provide seamless querying on the structured and unstructured data. Querying each kind of data in isolation has been the main subject of study for the fields of databases and information retrieval. Recently the database community has studied the problem of answering keyword queries on structured data such as relational data or XML data. The only combination that has not been fully explored is answering structured queries on unstructured data.

This paper explores an approach in which we carefully construct a keyword query from a given structured query, and submit the query to the underlying engine (e.g., a web-search engine) for querying unstructured data. We take the first step towards extracting keywords from structured queries even without domain knowledge and propose several directions we can explore to improve keyword extraction when domain knowledge exists. The experimental results show that our algorithm works fairly well for a large number of datasets from various domains.

1 Introduction

Significant interest has arisen recently in combining techniques from data management and information retrieval [1, 5]. This is due to the growing number of applications that require access to both structured and unstructured data. Examples of such applications include data management in enterprises and government agencies, management of personal information on the desktop, and management of digital libraries and scientific data. Such collections of data have been referred to as *dataspaces* [8], and *Dataspace Support Platforms (DSSPs)* were proposed to offer several services over dataspaces, including search and query, source discovery and categorization, indexing and some forms of recovery.

One of the key services of a DSSP is to provide seamless querying on the structured and unstructured data. Querying each kind of data in isolation has been the main subject of study for the fields of databases and information retrieval. Recently the database community has studied the problem of answering keyword queries on structured data such as relational data or XML data [10, 2, 4, 21, 11].

The only combination that has not been fully explored is answering structured queries on unstructured data. Informat-ion-extraction techniques attempt to extract structure from unstructured data such that structured queries can be applied. However, such techniques rely on the *existence* of some underlying structure, so are limited especially in heterogeneous environments.

This paper explores an approach in which we carefully construct a keyword query from a given structured query, and submit the query to the underlying engine (e.g., a web-search engine) for querying unstructured data. We begin by describing the many ways our strategy can contribute in supporting query answering in a dataspace.

1.1 Motivation

Broadly, our techniques apply in any context in which a user is querying a structured data source, whereas there are also unstructured sources that may be related. The user may want the structured query to be *expanded* to include the unstructured sources that have relevant information.

Our work is done in the context of the Semex Personal Information Management (PIM) System [6]. The goal of SEMEX is to offer easy access to all information on one's desktop, with possible extension to mobile devices, imported databases, and the Web. The various types of data on one's desktop, such as emails and contacts, Latex and Bibtex files, PDF files, Word documents and Powerpoint presentations, and cached webpages, form the major data sources managed by SEMEX. On one hand, SEMEX extracts instances and associations from these files by analyzing the data formats, and creates a database. For example, from LATEX and BIBTEX files, it extracts Paper, Person, Conference, Journal instances and authoredBy, publishedIn associations. On the other hand, these files contain rich text and SEMEX considers them also as unstructured data.

SEMEX supports keyword search by returning the instances whose attributes contain the given keywords and the documents that contain the keywords. In addition, SEMEX allows sophisticated users to compose structured queries to describe more complex information needs. In particular, SEMEX provides a graphical user interface to help users compose *triple queries*, which in spirit are the same as SPARQL queries [19] and describe the desired instances using a set of triples. Below is an example triple query asking for the papers that cite Halevy's SEMEX papers (note that users enter the queries through the user interface and never see the syntax below):

```
SELECT $t
FROM   $pa1 as paper, $pa2 as paper, $pe as person
WHERE  $pa1 cite $pa2, $pa2 title ‘‘Semex’’
       $pa2 author $pe, $pe name ‘‘Halevy’’
       $pa1 title $t
```

Ideally, the query engine should answer this type of queries not only on the database, but also on the unstructured data repository. For example, it should be able to retrieve the PDF or Word documents that cite Halevy's SEMEX papers, and retrieve the emails that mention such citations. More broadly, it should be able to search the Web and find such papers that do not reside in the personal data. The results can be given as file descriptions of the documents or links of the webpages, so the user can explore them further. This is the first place keyword extraction is useful: we can extract a keyword set from the triple query, and perform keyword search on the unstructured data repository and on the web.

Continuing with this example, suppose the user has imported several databases from external data sources, including the XML data from the Citeseer repository, and the technical-report data from the department relational database. When the user poses this triple query, she also expects results to be retrieved from the imported data. Note that although the imported data are structured, they do not share the same schema with the database created by SEMEX. In addition,

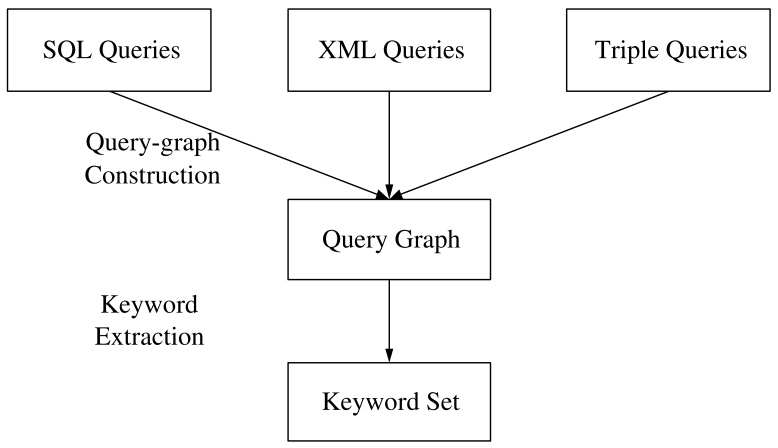

Figure 1: Our approach to keyword extraction. In the first step we construct a query graph for the structured input query, and in the second step we choose the node labels and edge labels from the graph that best summarize the query.

the mappings between the schemas are not given and typically cannot be easily established. One possible solution is to transform the query into keyword search by keyword extraction. Then, by applying existing techniques on answering keyword queries on structured data, SEMEX can retrieve relevant information even without schema mapping.

1.2 Our Contributions

In this paper, we study how to extract keywords from a structured query, such that searching the keywords on an unstructured data repository obtains the most relevant answers. The goal is to obtain reasonably precise answers even without domain knowledge, and improve the precision if knowledge of the schema and the structured data is available.

As depicted in Figure 1, the key element in our solution is to construct a *query graph* that captures the essence of the structured query, such as the object instances mentioned in the query, the attributes of these instances, and the associations between these instances. With this query graph, we can ignore syntactic aspects of the query, and distinguish the query elements that convey different concepts. The keyword set is selected from the node and edge labels of the graph.

Our algorithm selects attribute values and schema elements that appear in the query (they also appear as node labels and edge labels in the query graph so are referred to as *labels*), and uses them as keywords to the search engine. When selecting the labels, we wish to include only *necessary* ones, so keyword search returns exactly the query results and excludes irrelevant documents. We base our selection on the *informativeness* and *representativeness* of a label: the former measures the amount of information provided by the label, and the latter is the complement of the distraction that can be introduced by the label. Given a query, we use its query graph to model the effect of a selected label on the informativeness of the rest of the labels. By applying a greedy algorithm, we select the labels with the highest informativeness and representativeness.

In particular, our contributions are the following:

1. We propose a novel strategy to answering structured queries on unstructured data. We extract keywords from structured queries and then perform keyword search using information retrieval.
2. We take a first step towards extracting keywords from structured queries even without domain knowledge, and propose several directions we can explore to improve keyword extraction when

domain knowledge exists. The experimental results show that our algorithm works fairly well for a large number of datasets from various domains.

This paper is organized as follows. Section 2 discusses related work. Section 3 defines the problem. Section 4 describes our algorithm in selecting keywords from a given query. Finally, Section 5 presents experimental results and Section 6 concludes.

2 Related Work

The Database community has recently considered how to answer keyword queries on RDB data [10, 2, 4] and on XML data [21, 11]. In this paper, we consider the reverse direction, answering structured queries on unstructured data.

There are two bodies of research related to our work: the information-extraction approach and the query-transformation approach. Most information-extraction work [9, 16, 17, 18, 13, 7, 3] uses supervised learning, which is hard to scale to data in a large number of domains and apply to the case where the query schema is unknown beforehand.

To the best of our knowledge, there is only one work, SCORE [15], considering transforming structured queries into keyword search. SCORE extracts keywords from query results on structured data and uses them to submit keyword queries that retrieve supplementary information. Our approach extracts keywords from the query itself. It is generic in that we aim to provide reasonable results even without the presence of structured data and domain knowledge; however, the technique used in SCORE can serve as a supplement to our approach.

3 Problem Definition

We define the *keyword extraction* problem as follows. Given a structured query (in SQL, XQuery, etc.), we extract a set of keywords from the query. These keywords are used to construct a keyword query that returns information potentially relevant to the structured query. A keyword search on a large volume of unstructured data often returns many results; thus, we measure the quality of the answers using top-k precision — the percentage of relevant results in the top-k results. We consider queries that do not contain disjunctions, comparison predicates (e.g., $\neq$, $<$) or aggregation. Such queries are common in applications such as PIM and digital libraries.

The following example shows some of the challenges we face.

Example 1. *Consider a simple SQL query that asks for Dataspaces papers published in 2005.*

```
SELECT title
FROM   paper
WHERE  title LIKE '%Dataspaces%' AND year = '2005'
```

We have many options in keyword extraction and the following list gives a few:

1. *Use the whole query: "select title from paper where title LIKE 'dataspaces' and year = '2005' ".*
2. *Use the terms in the query excluding syntactic sugar (e.g.,* select, from, where*): "paper title +dataspaces year +2005". (Most search engines adopt the keyword-search syntax that requires the keyword following a "+" sign to occur in the returned documents or webpages.)*

3. Use only the text values: "+dataspaces +2005".

4. Use a subset of terms in the query: "+dataspaces +2005 paper title".

5. Use another subset of terms in the query: "+dataspaces +2005 paper".

A human would most probably choose the last keyword set, which best summarizes the objects we are looking for. Indeed, at the time of the experiment, Google, Yahoo, and MSN all obtained the best results on the last keyword set (Google obtained 0.6 top-10 precision), and the first hits all mentioned the dataspaces paper authored by Franklin et al. in 2005 (two of the search engines returned exactly the paper as the top hit). In contrast, for the first two keyword sets, none of the three search engines returned relevant results. For the third and fourth keyword sets, although some of the top-10 results were relevant, the top-10 precisions were quite low (Google obtained 0.2 top-10 precisions on both keyword sets). □

To explain the above results, we consider the possible effects of a keyword. On the one hand, it may narrow down the search space by requiring the returned documents to contain the keyword. On the other hand, it may distract the search engine by bringing in irrelevant documents that by chance contain the keyword. Ideally, we should choose keywords that are *informative* and *representative*: the keywords should significantly narrow down the search space without distracting the search engine. We describe our keyword-extraction approach in the next two sections.

4 Constructing keyword queries

To create a keyword query from a structured query Q, we first construct Q's query graph, and then extract keywords from it. Intuitively, the query graph captures the essence of the query and already removes irrelevant syntactic symbols. In this section, we first define the query graph and then describe keyword extraction.

4.1 Query Graph

In constructing the graph, we model the data as a set of object *instances* and *associations* between the instances. Each instance belongs to a particular *class* and corresponds to a real-world object. An instance is described by a set of *attributes*, the values of which are ground values. An association is a relationship between two instances. We can view a query as a subgraph pattern describing the queried instances with their attributes and directly or indirectly associated instances.

Definition 4.1. *(Query Graph) A* query graph $G_Q = (V, E)$ *for query* Q *is an undirected graph describing the instances and associations mentioned in* Q.

- *For each instance mentioned in* Q*, there is an* instance node *in* V*, labelled with the name of the instance class.*
- *For each association mentioned in* Q*, there is an* association edge *in* E *labelled with the association name. The association edge connects the two instance nodes involved in the association.*
- *For each ground value in* Q*, there is a* value node *in* V *labelled with the value, and an* attribute edge *in* E *labelled with the attribute name. The attribute edge connects the value node and the node of the owner instance.*

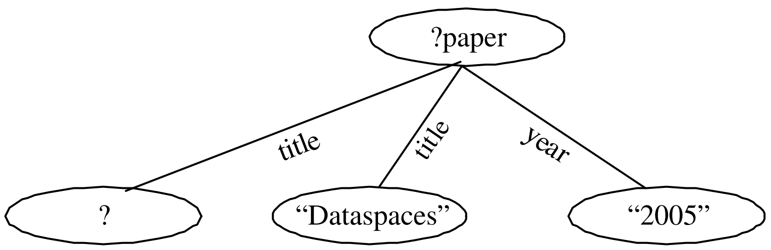

Figure 2: The query graph for the query in Example 1. The graph contains one instance node — paper, two value nodes — "Dataspaces" and "2005", and one question node. The nodes are connected by three attribute edges.

- *For each queried attribute in Q, there is a* question node *in V labelled with "?", and an attribute edge in E labelled with the attribute name. The attribute edge connects the question node and the node of the* queried instance. □

As an example, Figure 2 shows the query graph for Example 1.

4.1.1 Query-graph Construction

It is straightforward to construct query graphs for triple queries. The process is more tricky for SQL queries; for example, a table in a relational database can either record a set of object instances or a set of associations. We now describe in details how we construct a query graph for SQL queries.

Intuitively, attributes in a SELECT-clause correspond to question nodes. In the WHERE-clause, select predicates (in form of *attr* = *value* or *attr* LIKE *value*) correspond to value nodes, and join predicates correspond to association edges. Tables in a FROM-clause can correspond to either instance nodes or association edges.

We construct a query graph for a SQL query in two steps. In the first step, we construct a preliminary graph by considering all tables in the query as object instances. In the second step, we compact the graph by updating certain instances to association edges.

Specifically, the first step proceeds as follows:

- For each table in the FROM-clause, there is an instance node labeled with the table name.
- For each attribute in the SELECT-clause, there is a question node connected with the corresponding instance node. The edge between the question node and the instance node is an attribute edge labeled with the attribute name.
- For each select predicate in the WHERE-clause, there is a value node labeled with the given value, connected with the corresponding instance node. The edge between the value node and the instance node is an attribute edge labeled with the attribute name.
- For each join predicate in the WHERE-clause, there is an association edge connecting the two corresponding instance nodes, labeled with the two attribute names. We omit common terms such as "ID" and "key".

In the second step, we compact the graph by removing unnecessary instance nodes. Suppose a sequence of instance nodes $N_0, \ldots, N_t$ forms a chain, and each of $N_i, i \in [1, t-1]$ has only two neighbors N_{i-1} and N_{i+1}. We remove $N_1, \ldots, N_{t-1}$ and connect N_0 and N_t with an association edge, labeled with all labels of the nodes and edges on the path from N_0 to N_t. Note that if there are multiple nodes with the same label, we either remove all of them or leave all as instance nodes to keep consistency.

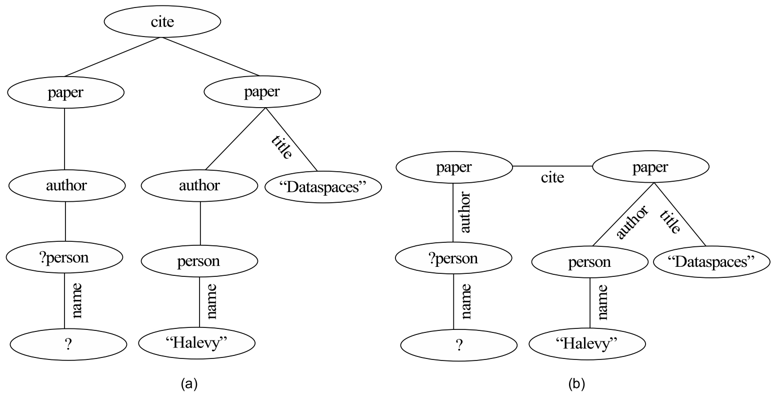

Figure 3: Constructing the query graph for the SQL query in Example 2. (a) In the first step, we construct a preliminary graph. (b) In the second step, we remove the cite node and the two author nodes.

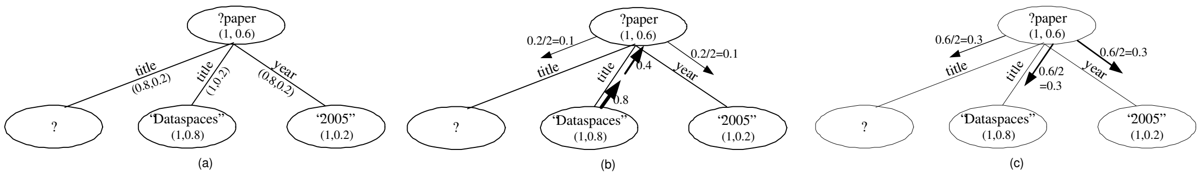

Figure 4: The query graph with i-scores and r-scores for the query in Example 1. (a) The initial (i-score, r-score) pairs. (b) The information flow representing the effect of the "Dataspaces" label. (b) The information flow representing the effect of the "Paper" label.

The above graph-construction algorithm assumes no knowledge of the query schema. However, in presence of such knowledge, we can refine the compacting step. For example, if we know that all attributes of a table T are foreign keys to other tables, we consider T as describing associations and shortcut T's neighbors with an association edge. To construct a query graph from an XML query is similar.

Example 2. *Consider the following SQL query:*

```
SELECT p1.name
FROM   Paper AS a1, Paper AS a2, Cite,
       Person AS p1, Person AS p2,
       AuthoredBy AS b1, AuthoredBy AS b2
WHERE  b1.pid = p1.id AND b1.aid = a1.id
  AND  b2.pid = p2.id AND b2.aid = a2.id
  AND  Cite.pid = a1.id AND Cite.cid = a2.id
  AND  p2.name LIKE '%Halevy%'
  AND  a2.title LIKE '%Semex%'
```

In the first step, we construct a graph shown in Figure 3(a). Note that the association edges do not have labels because all the involved attributes are various forms of "id". In the second step, we remove the cite *node and the two* author *nodes, obtaining the graph shown in Figure 3(b). We do not remove the* paper *nodes because there exists a* paper *node with value-node neighbors.* □

4.2 Extracting Keywords

We wish to include only *necessary* keywords rather than adding all relevant ones. This principle is based on two observations. First, a keyword often introduces distraction, so unnecessary keywords often lower the search quality by returning irrelevant documents. Second, real-world documents are often crisp in describing instances. For example, rather than saying "a paper authored by a person with name Halevy", we often say "Halevy's paper". Involving "authored by", "person" and "name" in the keyword set does not add much more information. We base our label selection on judging the informativeness and representativeness of labels. We first introduce measures for these two characteristics, and then describe our algorithm.

4.2.1 Informativeness and representativeness

Intuitively, informativeness measures the amount of information provided by a label term. For example, attribute values are more informative than structure terms. Representativeness roughly corresponds to the probability that searching the given term returns documents or webpages in the queried domain. For example, the term "paper" is more representative than the term "title" for the publication domain. We use *i-score* to measure informativeness and *r-score* to measure representativeness. Given a node label or edge label l, we denote its i-score as i_l, and r-score as r_l. Both i_l and r_l range from 0 to 1. Note that the representativeness of label l is the complement of l's *distractiveness*, denoted as d_l, so $d_l = 1 - r_l$. Figure 4(a) shows the initial (i-score,r-score) pair for each label (we will discuss shortly how we initialize the scores).

We observe that the informativeness of a label also depends on the already selected keywords. For example, consider searching a paper instance. The term "paper" is informative if we know nothing else about the paper, but its informativeness decreases if we know the paper is about "dataspaces", and further decreases if we also know the paper is by "Halevy". In other words, in a query graph, once we select a label into the keyword set, the informativeness of other labels is reduced.

We model the effect of a selected label s on the i-scores of other labels as an information flow, which has the following three characteristics:

- At the source node (or edge), the flow has volume r_s. The reason is that the effect of s is limited to the search results that are related to the queried domain, and this percentage is r_s (by definition).
- The information flow first goes to the neighbor edges or nodes (not including the one from which the flow comes). If s is a label of an instance node, the flow value is divided among the neighbor edges. Specifically, let n be the number of different labels of the neighbor edges, the flow volume on each edge is r_s/n. The division observes the intuition that the more distinct edges, the more information each edge label provides even in presence of the s label, and thus the less effect s has on these labels.
- After a flow reaches a label, its volume decreases by half. It then continues flowing to the nodes (or edges) at the next hop and is divided again, until reaching value nodes or question nodes. In other words, s's effect dwindles exponentially in the number of hops. Note that the flow is only affected by the r-score of the source node, but not the r-scores of other nodes that it reaches.

When we add a new label to the keyword set, we compute the effect of the label on the rest of the labels and update their i-scores. Once a keyword set is fixed, the i-scores of the rest of the labels are fixed, independent of the order we select the keywords. Figure 5 gives the formal algorithm for i-score update.

Example 3. *Consider the query graph in Figure 4(a). Figure 4(b) shows the effect of the value label "Dataspaces" on the i-scores of the rest of the nodes, and Figure 4(c) shows the effect of the*

```
procedure UpdateIScore($\mathcal{G}, S, r_S, I$)
//$\mathcal{G}$ is the input query graph;
//$S$ is the label just added to the keyword set
//$r_S$ is the r-score for the $S$ label
//$I$ is the array of i-scores for labels in $\mathcal{G}$;
  $queue = \{S\}$;
  $I[S] = r_S, P[S] = \{S\}$;
  while $queue \neq \emptyset$
    $T = pop(queue)$
    if $T$ is an edge label
      $f = 1$
    else
      $f = \#$(T's neighbor edges not including $P(S)$
        and with different labels
    for each label $L$ in $T$'s neighbors in $\mathcal{G}$
      if $L \in P(S) || r_L \leq 0$
        continue;
      $i_L - = I[T]/2f$;
      Add $L$ to $P[S]$;
      if $L \notin queue$
        push($queue, L$);
```

Figure 5: Algorithm for i-score update.

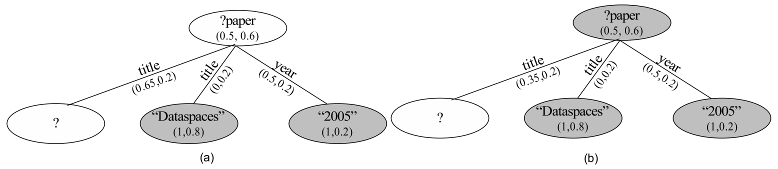

Figure 6: Extracting keywords from query graph in Figure 4(a). (a) The i-scores of the labels after selecting the labels "Dataspaces" and "2005". (b) The i-scores of the labels after selecting the label "Paper".

instance label "Paper". Note that we divide 0.6 by 2 rather than by 3, because the three edges are labelled by only two distinct labels. □

4.2.2 Selecting labels

When we select node or edge labels, we wish to choose those whose provided information is larger than the possible distraction; that is, $i > d = 1 - r$, so $i + r > 1$. We select labels in a greedy fashion: in each step we choose the label with the highest $i + r$, and terminate when there are no more labels with $i + r > 1$. Specifically, we proceed in three steps.

1. We choose all labels of value nodes. After adding each label to the keyword set, we update the i-scores of the rest of the nodes.
2. If there are labels satisfying $i + r > 1$, we choose the one with the largest $i + r$. We add the label to the keyword set and update the i-scores of the rest of the nodes.
3. We iterate step 2 until no more labels can be added.

Figure 7 gives the algorithm for label selection.

Example 4. *Consider the query graph in Figure 4(a). We select labels in two steps. In the first step, we select the labels of all value nodes, "Dataspaces" and "2005". The updated i-scores are*

```
procedure LabelSelection($\mathcal{G}, I, R$) return K
//$\mathcal{G}$ is the input query graph;
//$I$ is the array of i-scores for labels in $\mathcal{G}$;
//$R$ is the array of r-scores for labels in $\mathcal{G}$;
//$K$ is the selected keyword set;
  $K = \emptyset$;
  for each attribute-value label $V$ in $\mathcal{G}$;
    Add $V$ to $K$;
    UpdateIScore($\mathcal{G}, V, r_V, I$);
  while (true)
    Select the label L with the max $I(L) + R(L)$;
    if $I(L) + R(L) <= 1$ break;
    Add $L$ to $K$;
    UpdateIScore($\mathcal{G}, L, r_L, I$);
  return $K$;
```

Figure 7: Algorithm for label selection.

shown in Figure 6(a). We then select label Paper*, and the updated i-scores are shown in Figure 6(b). After this step no more labels satisfy the condition $i+r > 1$ so the algorithm terminates. The result keyword set is thus "Dataspaces 2005 paper".* □

4.2.3 Initializing i-scores and r-scores

We now discuss how to initialize the i-scores and r-scores. When we have no domain knowledge, we assign default values for different types of labels. We observe the web data for the representativeness of different types of nodes, and assign r-scores accordingly. For i-scores, we consider values and the class name of the queried instance as more informative and set the i-scores to 1, and consider other labels less informative. We will discuss the default score setting in our experiments in Section 5.

There are several ways to obtain more meaningful r-scores in presence of domain knowledge, and here we suggest a few. The first method is to do keyword search on the labels. Specifically, for a label l, we search l using the unstructured dataset on which we will perform keyword search. We manually examine the top-k (e.g., $k = 10$) results and count how many are related to the queried domain. The percentage λ is considered as the r-score for the l label.

Another approach is to do Naive-Bayes learning on a corpus of schemas and structured data in the spirit of [12]. As an example, we discuss how to compute the r-scores of instance names. We divide the schemas into a set of domains, each containing a set of schemas. Suppose the corpus contains domains $D_1, \ldots, D_l$, schemas $S_1, \ldots, S_m$, and class names $C_1, \ldots, C_n$. We say $S_j \in D_i$ if the schema S_j belongs to the domain D_i, and we say $C_k \in S_j$ if the class name C_k occurs in the schema S_j. We now apply Naive Bayes learning to calculate the probability that the label C of an

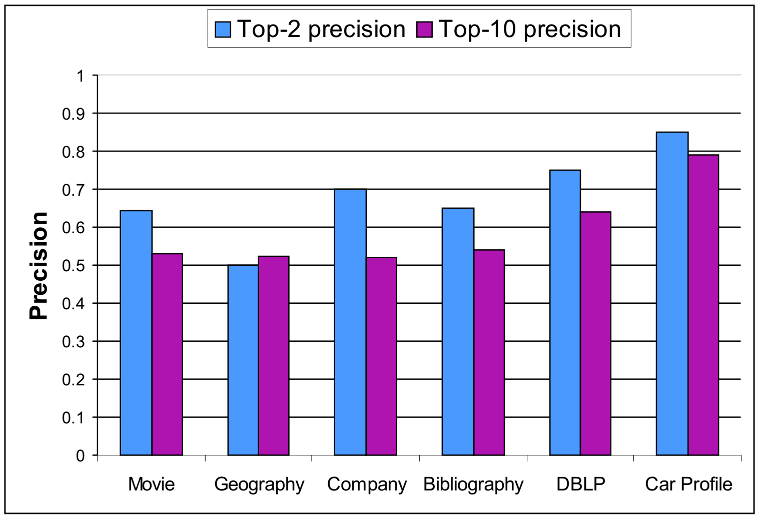

Figure 8: Top-2 and top-10 precision in different domains without applying domain knowledge. With our default settings of i-scores and r-scores, our algorithm performed well in all domains.

instance node represents a class in the queried domain D:

$$\begin{aligned}
P(D|C) &= \frac{P(C|D) \cdot P(D)}{P(C)} \\
&= \frac{\frac{|\{S_j|S_j \in D, C \in S_j\}|}{|\{S_j|S_j \in D\}|} \cdot \frac{|\{S_j|S_j \in D\}|}{m}}{\sum_{i=1}^{l} \left(\frac{|\{S_j|S_j \in D_i, C \in S_j\}|}{|\{S_j|S_j \in D_i\}|} \cdot \frac{|\{S_j|S_j \in D_i\}|}{m} \right)} \\
&= \frac{|\{S_j|S_j \in D, C \in S_j\}|}{\sum_{i=1}^{l} |\{S_j|S_j \in D_i, C \in S_j\}|} \\
&= \frac{|\{S_j|S_j \in D, C \in S_j\}|}{\sum_{i=1}^{m} |\{S_j|C \in S_j\}|}
\end{aligned}$$

The value of $P(D|C)$ can be considered as the r-score of the C label. Similarly, we can compute the r-scores for attribute names and association names. Finally, given an attribute a, to decide the r-score of its value labels, we randomly sample a number of values of the a attribute from the structured data and calculate the probability that the value belongs to the queried domain D. We use the average probability as the r-score.

The second approach learns the scores from the corpus, and so performs well only if the corpus and the unstructured data observe the same pattern. However, it is an alternative when the unstructured data source is unknown. Note that although this training phase is expensive, it is a one-time process and can significantly improve search performance.

5 Experimental Results

This section describes a set of experiments that begin to validate our keyword-extraction algorithm. Our goal is to show that our algorithm performs well even without domain knowledge, and that search quality improves when domain knowledge exists.

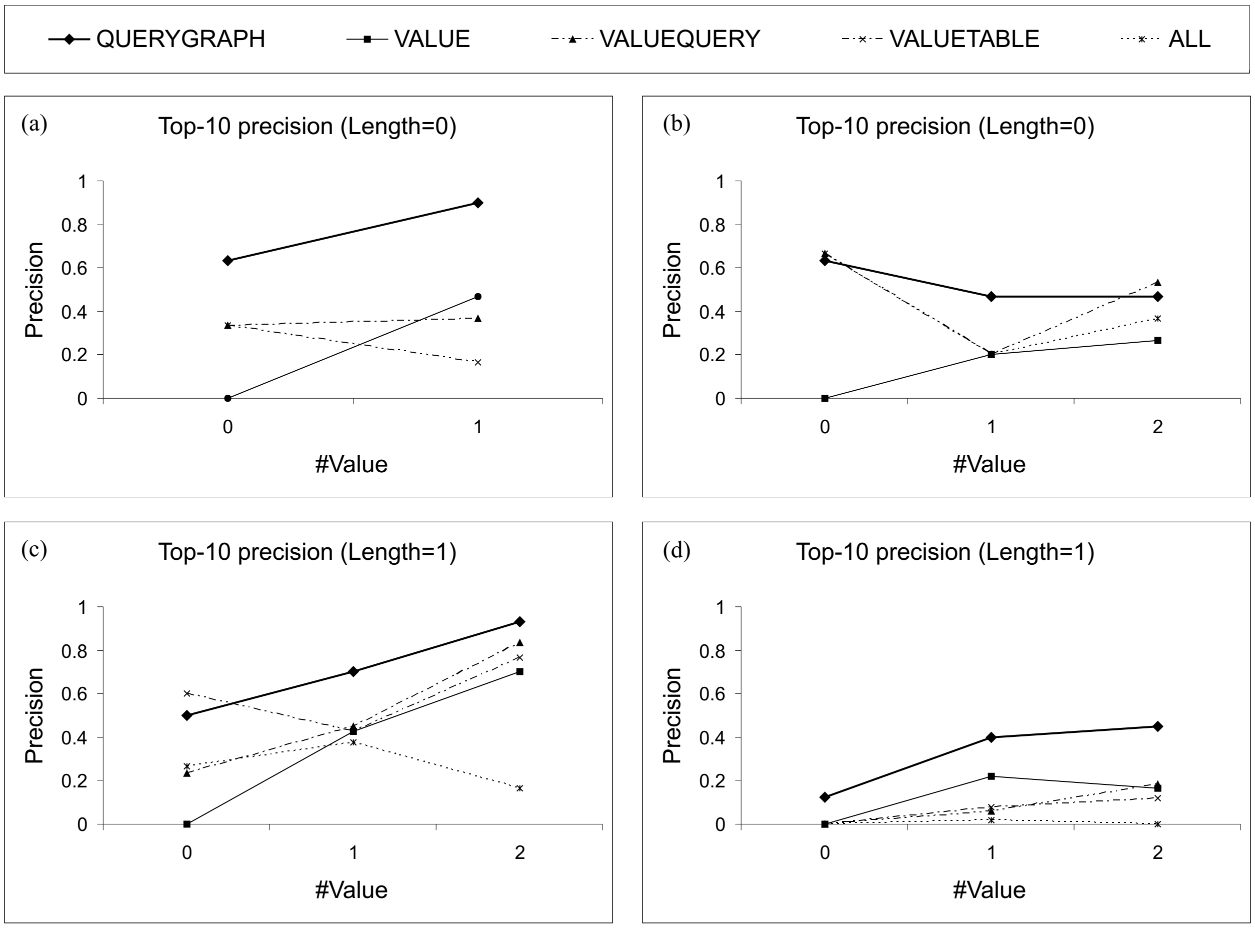

Figure 9: Top-10 precision for queries with length 0 in (a) the movie domain and (b) the geography domain, and with length 1 in (c) the movie domain, and (d) the geography domain. QueryGraph beat other solutions in most cases and the top-10 precision increased with the growing number of attribute values. (In (a) and (b) the ValueTable line and the QueryGraph line overlap, as the two methods extract the same keywords.)

5.1 Experiment Setup

We selected six different domains from the UW XML repository [20] and the Niagara XML repository [14], including movie, geography, company profiles, bibliography, DBLP, and car profiles. The schemas for these domains vary in complexity, such as the number of elements and attributes, and the number of children of each element.

When we selected queries, we varied two parameters in the selected queries: *#values* and *length*. The former is the number of attribute values in the query, indicating the amount of value information given by the query. The latter is the longest path from a queried instance (the instance whose attributes are queried) to other instances in the query graph, corresponding to the complexity of the structure information presented in the query. Finally, we randomly selected text values from the XML data for our queries. After generating the keyword set from the input queries, we used the Google Web API to search the web.

We measured the quality of our extracted keywords by top-k precision, which computes the percentage of the top k hits that provide information relevant to the query. We analyzed the results using top-2 and top-10 precision.

Finally, we set the default values for i-scores and r-scores as follows (we used the same setting

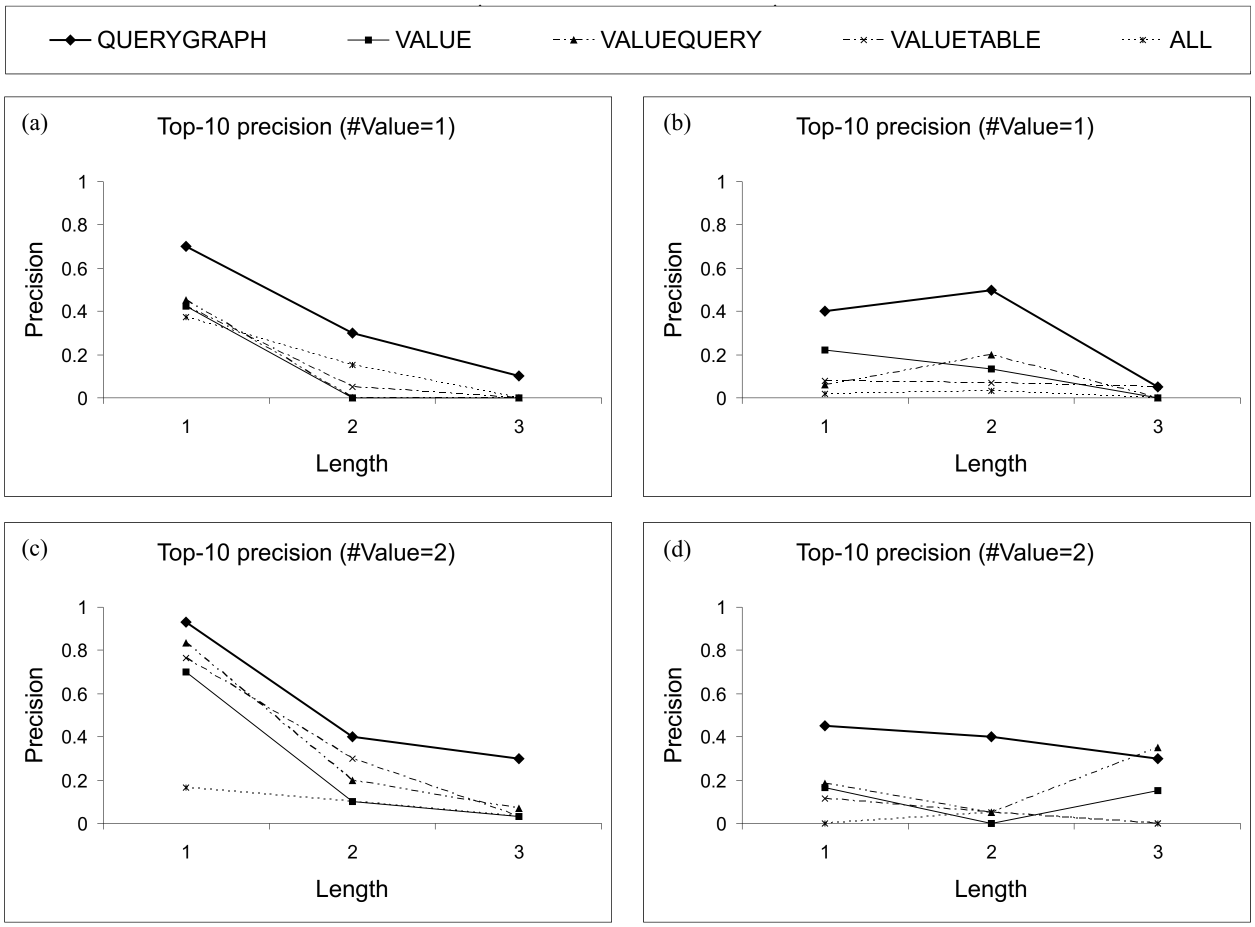

Figure 10: Top-10 precision of queries with one attribute value in (a) the movie domain and (b) the geography domain, and with two attribute values in (c) the movie domain and (d) the geography domain. QueryGraph beat other solutions in most cases and the top-10 precision went down as the query length increased.

for all domains).

- **i-scores:** 1 for value labels and labels of queried instances, and 0.8 for other labels.
- **r-scores:** 0.8 for text-value labels and labels of associations between instances of the same type, 0.6 for instance labels, 0.4 for association labels, 0.2 for attribute labels, and 0 for number-value labels.

5.2 Experimental Results

We validated our algorithm on six domains, and the results are shown in Figure 8. We observe that our algorithm performed well in all domains. With our default settings for i-scores and r-scores, the top-2 and top-10 precisions in different domains were similar. The average top-2 precision was 0.68 and the average top-10 precision was 0.59.

5.2.1 Contributions of the Query Graph

We now compare QueryGraph with several other approaches that select terms directly from the query.

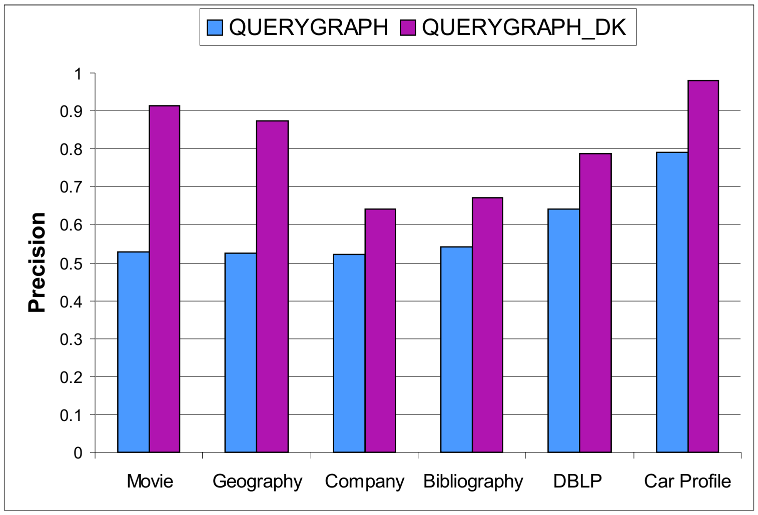

Figure 11: Top-10 precision. QueryGraph_DK applies domain knowledge and QueryGraph does not. It shows that by applying domain knowledge, our algorithm can further improve search quality.

- All: Include all terms except syntactic symbols.
- Value: Include only ground values.
- ValueQuery: Include ground values and all table and attribute names in the SELECT-clause.
- ValueTable: Include ground values and all table names in the FROM-clause.

We report the results on two domains: movie and geography. We observed similar trends on other domains.

Varying the number of values: We first consider the impact of #values on keyword extraction. We considered queries with length 0 or 1, and varied #values from 0 to 2 when it applies. Figure 9 shows the top-10 precision.

We observed the following. First, in most cases QueryGraph obtained higher precision than the other approaches. It shows that including appropriate structure terms obtained much better results than searching only the text values. Second, when the number of attribute values increases, most approaches obtained better search results, but All performed even worse because it includes distractive keywords.

Varying query length: We now examine the effect of the structure complexity on search performance. We considered queries with 1 or 2 attribute values, and varied the length from 1 to 3. Figure 10 shows the results. We observed that our algorithm again beat other methods in most cases. As the length of the query grew, the top-10 precision dropped. This is not a surprise as complex query structure complicates the meaning of the query.

5.2.2 Applying Domain Knowledge

We finally examine how the domain knowledge helps in keyword extraction. When we applied domain knowledge, the average top-2 precision was 0.92 and the average top-10 precision was 0.81. Figure 11 shows a comparison of top-10 precisions with and without domain knowledge on various domains. The top-10 precisions increased 39% on average. It shows that our algorithm can further improve the search quality by applying domain knowledge.

6 Conclusions and Future Work

We described an approach for extracting keyword queries from structured queries. The extracted keyword queries can be posed over a collection of unstructured data in order to obtain additional data that may be relevant to the structured query. The ability to widen queries in this way is an important capability in querying dataspaces, that include heterogeneous collections of structured and unstructured data.

Although our experimental results already show that our algorithm obtains good results in various domains, there are multiple directions for future work. First, we can refine our extracted keyword set by considering the schema or maybe even a corpus of schemas. For example, we can replace an extracted keyword with a more domain-specific keyword in the schema; we can also add keywords selected from the corpus to further narrow down the search space. Second, we can use existing structured data, as proposed in SCORE [15], to supplement the selected keyword set. Third, we can perform some linguistic analysis of the words in the structured query to determine whether they are likely to be useful in keyword queries. Finally, we would like to develop methods for ranking answers that are obtained from structured and unstructured data sources.

References

[1] S. Abiteboul and et al. The Lowell database research self assessment. *CACM*, 48(5), 2005.

[2] S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: A system for keyword-based search over relational databases. In *ICDE*, 2002.

[3] R. Baumgartner, S. Flesca, and G. Gottlob. Visual web information extraction with lixto. In *VLDB*, 2001.

[4] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. In *ICDE*, 2002.

[5] S. Chaudhuri, R. Ramakrishnan, and G. Weikum. Integrating DB and IR technologies: What is the sound of one hand clapping? In *CIDR*, 2005.

[6] X. Dong and A. Halevy. A platform for personal information management and integration. In *CIDR*, 2005.

[7] O. Etzioni, M. Cafarella, and D. Downey. Web-scale information extraction in KnowItAll (preliminary results). In *Proc. of the Int. WWW Conf.*, 2004.

[8] M. Franklin, A. Halevy, and D. Maier. From databases to dataspaces: A new abstraction for information management. *Sigmod Record*, 34(4):27–33, 2005.

[9] D. Freitag and A. McCallum. Information extraction with HMMs and shrinkage. In *AAAI-99 Workshop on Machine Learning for Information Extraction*, 1999.

[10] V. Hristidis and Y. Papakonstantinou. Discover: Keyword search in relational databases. In *VLDB*, 2002.

[11] V. Hristidis, Y. Papakonstantinou, and A. Balmin. Keyword proximity search on XML graphs. In *ICDE*, 2003.

[12] J. Madhavan, P. A. Bernstein, A. Doan, and A. Y. Halevy. Corpus-based schema matching. In *ICDE*, 2005.

[13] A. McCallum. Efficiently inducing features or conditional random fields. In *UAI*, 2003.

[14] Niagara XML repository. http://www.cs.wisc.edu/ niagara/data.html, 2004.

[15] P. Roy, M. Mohania, B. Bamba, and S. Raman. Towards automatic association of relevant unstructured content with structured query results. In *CIKM*, 2005.

[16] M. Skounakis, M. Craven, and S. Ray. Hierarchical hidden markov models for information extraction. In *IJCAI*, 2003.

[17] S. Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34(1-3):233–272, 1999.

[18] S. Soderland, D. Fisher, J. Aseltine, and W. Lehnert. Crystal: Inducing a conceptual dictionary. In *IJCAI*, 1995.

[19] SPARQL. http://www.w3.org/TR/rdf-sparql-query/, 2003.

[20] UW XML data repository. http://www.cs.washington.edu/ research/xmldatasets/, 2002.

[21] Y. Xu and Y. Papakonstantinou. Efficient keyword search for smallest LCAs in XML databases. In *Sigmod*, 2005.