Semantic Queries by Example

Lipyeow Lim
University of Hawai‘i at Mānoa
lipyeow@hawaii.edu

Haixun Wang
Microsoft Research Asia
haixunw@microsoft.com

Min Wang
Google, Inc.
minw83@gmail.com

ABSTRACT

With the ever increasing quantities of electronic data, there is a growing need to make sense out of the data. Many advanced database applications are beginning to support this need by integrating domain knowledge encoded as ontologies into queries over relational data. However, it is extremely difficult to express queries against graph structured ontology in the relational SQL query language or its extensions. Moreover, semantic queries are usually not precise, especially when data and its related ontology are complicated. Users often only have a vague notion of their information needs and are not able to specify queries precisely. In this paper, we address these challenges by introducing a novel method to support semantic queries in relational databases with ease. Instead of casting ontology into relational form and creating new language constructs to express such queries, we ask the user to provide a small number of examples that satisfy the query she has in mind. Using those examples as seeds, the system infers the exact query automatically, and the user is therefore shielded from the complexity of interfacing with the ontology. Our approach consists of three steps. In the first step, the user provides several examples that satisfy the query. In the second step, we use machine learning techniques to mine the semantics of the query from the given examples and related ontologies. Finally, we apply the query semantics on the data to generate the full query result. We also implement an optional active learning mechanism to find the query semantics accurately and quickly. Our experiments validate the effectiveness of our approach.

Categories and Subject Descriptors

H.2 [**Database Management**]: Query Processing; H.3.3 [**Information Search and Retrieval**]: Query formulation

General Terms

Algorithms

1. INTRODUCTION

A growing number of advanced applications such as product information management (PIM) systems, customer relationship management (CRM) systems, electronic medical records (EMRs) systems are recognizing the need to incorporate ontology into the realm of object relational databases (ORDBMs) so that the user can query data and its related ontology in a consistent manner [7, 14, 22]. However, it is extremely tedious and time consuming to understand ontology, and use ontology in database queries [15]. Such queries that leverage the semantic information stored in ontologies to filter and retrieve data from relational tables are called *semantic queries*. The success of the relational database technology is at least partly due to the spartan simplicity of its data model and query language, which insulate the user from the physical implementation of the database. But for semantic queries, users are often exposed to the full complexity of the ontology. Still, integrating data and its related ontology is a challenge too important to ignore. There are two major approaches in attacking this problem. One is to flatten graph-structured ontologies into relational form [7, 22], and the other is to extend ORDBMSs and SQL to handle non-relational data directly [13, 14]. However, both approaches incur tremendous system cost, but have limited success in taking the tediousness out of handling semantic queries. In many cases the complexity of expressing semantic queries is almost prohibitive to allow for widespread use of such query mechanisms.

The future success of incorporating ontologies into practical database query processing depends on whether we can find automatic or semi-automatic methods to help users express semantic queries. In this paper, we introduce a novel approach that insulates the users from the complexity of the ontology, yet still enables them to ask every possible semantic query. The semi-automatic framework we develop bridges the gap between a query in a user's mind and the final result of the query. Furthermore, since users do not handle the ontology directly, there is no need to map the ontology into relational form, which means our approach does not incur expensive cost of extending database engines and query languages.

Before we dive into the details of our approach, we use some examples to illustrate the task on our hand. Consider an EMR (electronic medical records) system. Clinicians recording the diagnosis of a patient visit may choose different disease codes for the same symptoms that the patient is exhibiting. One clinician might describe a patient diagnosis using the code for "Tumor of the Uvea", while an-

vID	date	patientID	diagnosis
1	20080201	3243	Brain Neoplasm
2	20080202	4722	Stomatitis
3	20080202	2973	Tumor of the Uvea
4	20080204	9437	Corneal Intraepithelial Neoplasia
5	20080205	2437	Choroid Tumor
...	...	...	...

Figure 1: The table `visit` recording patient visits

other might use the code for "Iris Neoplasm". In the patient's EMR a generic term like "Eye Neoplasm" might be recorded instead of the more specific "Tumor of the Uvea" (we will use the more descriptive terms instead of the corresponding codes in this paper in the interest of readability). Hence, in order to obtain meaningful results from querying an EMR database, the query processing system needs to understand the semantics of the query and the data.

The growing demand for processing data and its meaning has spurred the increasing use of ontology in various applications. Continuing with the EMR example, many ontologies have been developed to capture the semantics of various sub-components of EMR data. For example, the National Cancer Institute (NCI) Thesaurus [17] is a collection of ontologies spanning the following areas: Drugs and Chemicals, Diseases Disorders and Findings, Anatomic Structure System or Substance, Gene, Chemotherapy Regimen etc. Figure 2 shows a fragment of the NCI Thesaurus in graphical form. Moreover, many of these ontologies are well integrated with existing data coding terminologies (eg, SNOMED [10], ICD9 [9]) used by industry EMR formats such as HL7 [8] and CCR [5]. With the confluence of ontologies, coding terminologies, and data standards, the need for querying relational data together with its related ontology has become even more urgent.

However, there are two fundamental challenges in this task. First, it is extremely difficult to express queries against a graph structured ontology in SQL as the following example illustrates.

Example 1 (Running Example). *Suppose we have a table of patient visit records as shown in Figure 1, of which the diagnosis column is associated with the NCI Thesaurus ontology (Figure 2). Consider the query to find all patients diagnosed with eye tumor.*

Using existing RDF-like data models [22], we could store the ontology as triples in the `Thesaurus(src,rel,tgt)` relation and attempt to write the query in Example 1 using recursive SQL as follows.

```
WITH Traversed (src) AS (
  (SELECT src
  FROM Thesaurus
  WHERE tgt = 'Eye Tumor' AND rel='Synonym')
UNION ALL
  (SELECT CH.tgt
  FROM Traversed PR, Thesaurus CH
  WHERE PR.src = CH.src AND CH.rel='is_a'))
SELECT DISTINCT V.*
FROM Thesaurus T, Visit V
WHERE src IN
  (SELECT DISTINCT src FROM Traversed)
  AND T.rel = 'Synonym'
  AND T.tgt = V.diagnosis
```

Alternatively, if we write the same query against the original XML format of the NCI Thesaurus, we have the following.

```
WITH Traversed (cls,syn) AS (
    (SELECT R.cls, R.syn
    FROM XMLTABLE ('Document("Thesaurus.xml")
      /terminology/conceptDef/properties
      [property/name/text()="Synonym" and
      property/value/text()="Eye Tumor"]
      /property[name/text()="Synonym"]/value'
      COLUMNS
      cls CHAR(64) PATH './parent::*/parent::*
                          /parent::*/name',
      tgt CHAR(64) PATH'.') AS  R)
UNION ALL
    (SELECT CH.cls,CH.syn
    FROM Traversed PR,
      XMLTABLE ('Document("Thesaurus.xml")
      /terminology/conceptDef/definingConcepts/
      concept[./text()=$parent]/parent::*/parent::*/
      properties/property[name/text()="Synonym"]/value'
      PASSING  PR.cls AS "parent"
      COLUMNS
      cls CHAR(64) PATH './parent::*/
                        parent::*/parent::*/name',
      syn CHAR(64) PATH'.') AS CH))
SELECT DISTINCT V.*
FROM Visit V
WHERE V.diagnosis IN
  (SELECT DISTINCT syn FROM Traversed)
```

In both instances, it is not straight-forward to write the query and the user needs to have an intimate knowledge of the structure or schema of the ontology such as existence of "synonym", and "is_a" edges.

The second fundamental challenge is the inherent fuzziness in the semantics of the query. In most practical applications, the data and the ontology behind it are quite complicated and consequently the queries are no longer exact, that is, users often have no more than a vague notion, rather than a clear understanding and definition, of what they are looking for. In other words, even if the users have intimate knowledge of the structure of the ontology, they might not be able to precisely specify what they want to find.

Example 2. *Find all patients diagnosed with some disease in the choroid, which is part of the eye.*

Intuitively, the user wants to find patients with some disease that affects or is located in the choroid (Figure 3 shows one possible processing workflow for this query). In the NCI Thesaurus, there are 3 separate types of relationship linking disease concepts to anatomic locations:

1. Disease_Has_Primary_Anatomic_Site
2. Disease_Has_Associated_Anatomic_Site
3. Disease_Has_Metastatic_Anatomic_Site

Even if the user knows the structure of the NCI Thesaurus ontology, i.e., he knows about the three types of relationships that are relevant to the query, without looking at the results, the user still may not know whether the query should use one of these relationships to "choroid" or all of them. If the user does not know the structure of the ontology at all, then he certainly would not know how to specify the query exactly. The query semantics is certainly not crisp and hence not easily expressed in SQL especially when the structure of the ontology is not well-known to the user.

Our approach. In this paper, we present a novel method to address the two challenges we identified above. Instead of endeavoring to incorporate ontology into relational form and create new language constructs to express such queries, our

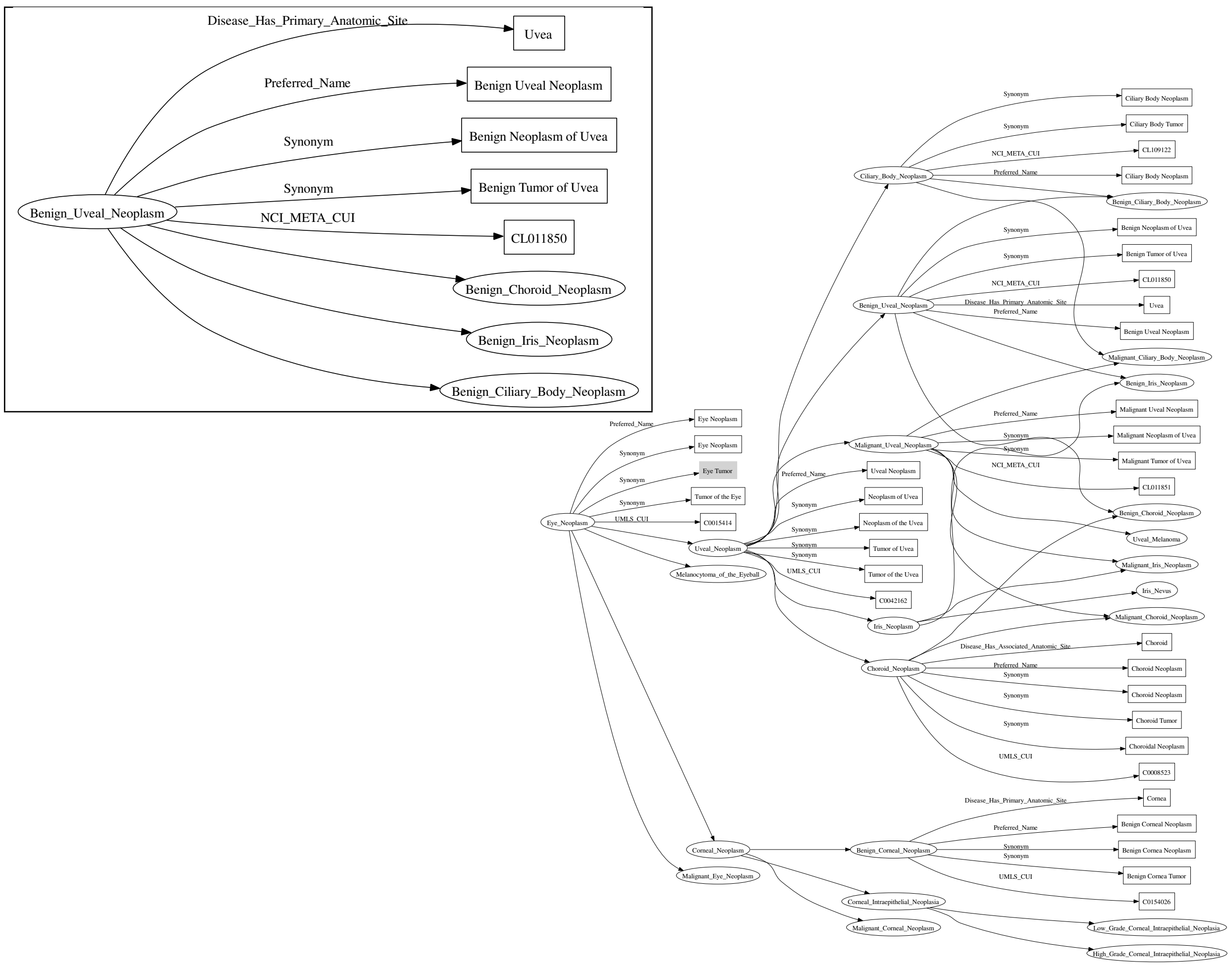


Figure 2: A three level fragment of the NCI Thesaurus ontology. Elliptical nodes denote concepts. Rectangular nodes denote property values. Edges between concepts denote subsumption relationships.

approach takes advantage of user-provided examples that satisfy the query as seeds to automate the query process.

Specifically, our method consists of three steps. In the first step, the user provides several examples that satisfy the query he has in mind. In the second step, the system uses machine learning techniques to mine the semantics of the query from the given examples and their related ontologies. In the third step, the system applies the semantics on the data to generate the entire query result and return it to the user. In the query processing process, as an optional active learning mechanism, the user will be probed systematically to determine whether certain tuples satisfy his query in order to help clarify his intention and speed up the query processing process. False positive errors in the query results can be detected by the user and fed back to the system as part of the active learning process. Since a semantic query is inherently fuzzy, the user typically expects only a subset of the full results, false negative errors can be ameliorated by doing active learning until the desired number of result tuples are obtained.

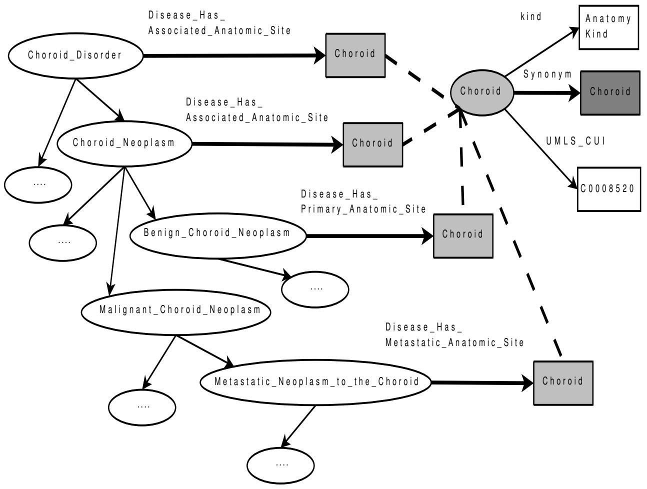
Using our method for Example 1, the user no longer needs to write the unwieldy SQL queries, but instead provides examples of tuples, say tuples with vID 4 & 5, that satisfy the query he has in mind. Our method then uses machine-learning techniques to learn a model from the examples, and generate the query results by applying the model on the entire base table. In other words, our method insulates the user entirely from the complexity of understanding and using ontologies. Consequently, there is no need to map ontologies into relational form.

2. RELATED WORK

Query by Example (QBE) is a well-known concept in database community. It was first proposed by Moshé M. Zloof in the mid 1970s [27, 28] as a query language that can be used by database users to define and query a relational database. It is quite different from SQL in that it is a graphical query language. Its interface is usually virtual tables where the user could enter commands, examples, etc. After QBE was presented, most research work around QBE has been focused on enrichment and extension of QBE as a query language and developing efficient methods for generating and processing the queries defined by the examples [12].

In commercial database products, QBE is widely used as graphical front-end for RDBMSs [1]. It is also used as a convenient interface for users to specify queries for image, video, and document databases, and various techniques have been studied [2, 3, 4, 20].

There are two common characteristics for all the previous



Disease_Has_ Associated_Anatomic_Site
Choroid_Disorder
Choroid
kind
Anatomy Kind
Synonym
Choroid
Choroid
Disease_Has_ Associated_Anatomic_Site
Choroid_Neoplasm
Choroid
UMLS_CUI
C0008520
....
Disease_Has_ Primary_Anatomic_Site
Benign_Choroid_Neoplasm
Choroid
Malignant_Choroid_Neoplasm
Disease_Has_ Metastatic_Anatomic_Site
Metastatic_Neoplasm_to_the_Choroid
Choroid

Figure 3: One possible processing workflow for Example 2 is to start from the user-given term "Choroid" (on extreme right), find the corresponding concept node, find all the disease nodes (on far left) that are linked to the Choroid concept via a "Disease_Has_Primary_Anatomic_Site" or "Disease_Has_Associated_Anatomic_Site" or "Disease_Has_Metastatic_Anatomic_Site" edge, and expand these disease nodes to find all subsumed concepts and their synonyms.

QBE work: (a) the examples are used to specify a query that will be generated (b) the generated query is a "normal" query in terms that all the query conditions (may be in the forms of similarity measures) are defined on the *base attributes* in the underlying tables in the databases.

The semantic QBE framework proposed in this paper is very different from the traditional QBE framework. First, due to the complexity of the semantic information associated with the data in the base relational tables, the real query associated with the user's intention which is specified by the input examples is really hard (or impossible) to capture by a traditional SQL query. As a result, our method tries to *learn the query processing directly without learning the query first*, in contrast to the traditional QBE framework. Secondly, in our work, the underlying "query" is defined not only on the base attributes in the relational table, but also on the semantics of the base data encoded in the ontology and the connections between the relational data and the ontology data.

It is worth mentioning the *Query By Semantic Example* (QBSE) work presented in [20]. While it looks very similar to our work in name, it still falls into the category of the traditional QBE work: the semantic information related to the images is in the form of a vocabulary that is used to train and label the base data during a off-line pre-processing stage. In other words, once the pre-processing is done, QBSE is not different from the traditional QBE.

Managing ontology data alone is not a new topic and several systems have been developed [16, 18, 19, 23, 24] during the past years. Some of these systems store ontology data in a file system, making querying them very hard [19]. The other systems transform the ontology data into RDF form and store the RDF triples in a relational database. Processing of ontology-related queries in these systems is typically done by an external middle-ware (wrapper) layer built on top of a DBMS engine, and DBMS users cannot really reference ontology data directly.

Querying relational data together with their semantics encoded in ontology is an emerging topic that has attracted a lot of attention recently. Das et al. [7] proposed a method to support ontology-based semantic matching in RDBMS using SQL directly. Ontology data are pre-processed and stored in a set of system-defined tables. Several special operators and a new indexing scheme are introduced. A database user can thus reference the ontology data directly using the new operators. The main drawback of this approach is that semantic queries involving the ontology data are usually hard to write and costly to process (in terms of both processing time and storage overhead) due to the graphical structure of the ontology data and the need for reasoning (i.e., transitive closure computation) on the ontology data.

In [14], *virtual view* is proposed as a way to represent relational data together with their related ontology data in a relational view. However, there are three requirements to apply the virtual view idea: (a)language extensions to SQL to support the creation and use of the virtual view, (b) the DBMS engine must support native XML data (together with relational data) and the processing of the virtual view related operators, and (c) to create a virtual view, the user must understand the complex ontology data and their relationship with the base relational data completely.

The problems and challenges of expressing semantic queries over relational data have also been described in [15]; however, the authors did not propose any concrete solution. This paper proposes a semantic query by example framework that addresses the challenges outline in [15].

3. SEMANTIC QUERY BY EXAMPLE

Suppose the semantic queries Q we consider are posed against a base table (or a view) D, and each record in D is of the following form:

$$(\mathbf{X}, \texttt{OID})$$

where $\mathbf{X}$ is a feature vector and `OID` is the ID of an ontology instance related to $\mathbf{X}$. For instance, $\mathbf{X}$ represents a patient, and `OID` is the disease code of the patient. In practice, D can be the outcome of a query, or a view, and each tuple of D can have multiple `OID` columns referring to multiple nodes in an ontology.

An important question is what semantic queries Q we can ask against D. Standard relational queries, which treat `OID` as a standard relational column, are unable to express the semantic structure in the ontology. Alternatively, we can employ non-relational languages such as XQuery and XPath to explore the semantic structure. However, it is extremely tedious to form a query.

Our method consists of an off-line phase and and online phase. In the off-line phase, the base table data is scanned once to compute the feature vector associated with the `OID` in each record. In the on-line phase, a user poses a query on the base table by giving a set of example tuples that satisfy the query he has in mind. The on-line phase uses a machine learning method, in this case, the support vector machine (SVM) method, to learn a model from the examples supplied by the user. Once the model is learned, the base tuple is scanned again, and the model decides what are the tuples satisfy the query. The user-provided training dataset may be of small size and/or low quality, which incur negative impact on the accuracy of the model. In this case, the user is probed systematically to determine whether predictions made by the system is accurate for an additional small number of tuples. We use a more advanced *active learning* algorithm for this purpose.

3.1 Feature Extraction

Before we can model a semantic query, we must first expand the base data to include related information in the ontology. In other words, for each record of $(\mathbf{X}, \texttt{OID}) \in D$, we need to expand it into

$$(\mathbf{X}, \mathbf{Y})$$

where $\mathbf{Y} = f(\texttt{OID})$ is a vector of features returned by a feature extraction function f. Our goal is to design the function f so that it captures enough ontological information related to the ontology object OID for modeling the query.

Challenges. In general learning problems, objects are represented as vectors in a feature space. For complex objects such as sequences, trees, and graphs, it is not a trivial task to find a suitable feature space: On the one hand, the feature space must capture enough information about the objects so that all discriminative features about an object are covered. However, in most cases, considering all possible features is prohibitive since it often leads to combinatorial explosion.

In our case, we want to vectorize the local hierarchical structure of an ontology node, as the local structure captures most semantic information of the node. Let o be the node whose id is OID in the ontology graph. The problem is, what information related to o in the ontology is important in classifying examples where OID is part of the data.

There are a large number of options: we can include o's immediate child nodes, immediate parent nodes, the paths to some nodes we consider important in the ontology, etc. These naive schemes are flawed for two reasons. First, the nodes and paths themselves are still complex structures, and for example we need to further vectorize the paths. Second, and more importantly, the approach is not general enough. As an instance, in Example 1, the critical information about the query consists of the synonyms of any descendant concepts of "Eye Tumor" – such information is not captured by the naive scheme mentioned above.

Alternatively, we can perform feature extraction after the query is given. For Example 2, we might want to include the information about whether there is a path to the anatomic site "Choroid" in the feature extraction process. But this approach is also flawed. First, feature extraction is an expensive process, and in most cases we cannot afford doing that for each query. Second, it undoes the purpose of query by example, as it requires the user to know what part of ontology is relevant and important in answering the query.

Shortest distance based feature extraction. In this paper, we adopt a simple and effective feature extraction method. Our approach is general. Although there is no guarantee that all queries can be answered accurately based on the features extracted by our approach, it endeavors to include as much information as possible while keeping the data size small and the extraction process simple.

More specifically, assume the ontology has a set of concepts $C = \{c_1, \cdots, c_k\}$. The feature extraction function we adopt is in the following form

$$f(\texttt{OID}) = \langle y_1, \cdots, y_k \rangle$$

where k is the total number of unique concepts in the ontology graph, and y_i is the shortest distance from node o (whose id is OID) to any node whose label or concept is c_i.

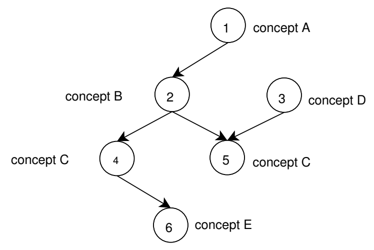


Figure 4: An ontology graph consisting of 5 concepts and 6 nodes

	A	B	C	D	E
1	0	1	2	3	3
2	1	0	1	2	2
3	3	2	1	0	4
4	2	1	0	3	1
5	2	1	0	1	3
6	3	2	1	4	0

Table 1: Extracted features

As an example, Figure 4 is a snippet of an ontology graph. Each node has its id and a label which indicates an ontological concept represented by the node. We omit the edge label here, as labeled edges can be transformed into labeled nodes. Table 1 shows the extracted features for the nodes. Here, each row represents a node and each column a concept in the ontology graph. An element f_{ij} in the table indicates the shortest distance between node i and concept j.

The size of the dataset is $n \cdot k$, where n is the total number of nodes and k is the total number of concepts. Given usually $k \ll n$, we know the total size is less than n^2. Unlike other graph feature extraction methods that result in information exponential in the size of the graph, this method is clearly more tractable.

Expressive power of the feature extraction method. The important question is whether the feature extraction method retains enough useful information about the local structure of each node. Clearly, it omits much information in the ontology. For instance, it treats the ontology as an undirected graph; it disregards the labels of the edges and concentrate on the labels of the nodes only.

Although it is not possible to rebuild the entire graph from the extracted data, we argue that the feature extraction method effectively captures the local structures of a node in the ontology. First of all, it tells us how close a node is to different concepts. Furthermore, we can infer much hierarchical information from the extracted features. The reason is because the distribution of different concepts in a graph is not uniformly random. In most cases, the concepts alone include rich hierarchical information. Typically, nodes belong to the same concept cluster together, or appear on the same hierarchical level in an ontology. Thus, from feature distributions of the nodes, we can roughly tell their hierarchical positions in the graph.

For example, assume a user wants to find nodes that are parent of any concept C nodes. Instead of expressing

	A	B	C	D	E	label
1	0	1	2	3	3	-1
2	1	0	1	2	2	+1
3	3	2	1	0	4	+1
4	2	1	0	3	1	-1
5	2	1	0	1	3	-1
6	3	2	1	4	0	-1

Table 2: Training dataset for query 'find nodes that are parent nodes of any node of concept C'

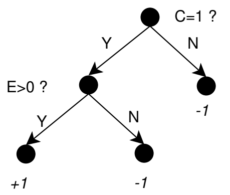


Figure 5: Decision tree classifier learned from the training dataset

the query explicitly, the user gives us the following training dataset, where node 2 and 3 are positive (satisfying the query), and the rest are negative. The question is, can we learn the query from the training dataset?

The answer is yes. A possible decision tree learned from the above data is shown in Figure 5. Given a node to classify, it first tests if the shortest distance from the node to concept C is 1. Nodes that satisfy the test are either parent or child nodes of concept C node, while other nodes are classified as negative immediately. To exclude child nodes of concept C node, the second test checks the shortest distance to concept E node. Because the distribution of concepts is not uniformly random, we have reason to believe that child nodes of concept C nodes are closer to concept E node than parent nodes. Thus, the decision tree learned should generalize well over testing data.

Certainly, it is highly possible that the user-provided training dataset, which is usually small, cannot fully characterize the query (for example, if the last record in Table 2 is not in the training dataset, then the decision tree will over generalize the query and classify both child and parent nodes of concept C nodes as positive). To make sure that the model we learn eventually has high accuracy, we resort to active learning, that is, we select records that are important to characterizing the query and ask the user to label these records in the learning process. We give the detail in Section 3.3.

Implementation. Algorithm 1 outlines the steps for the off-line phase. The OFFLINEPHASE procedure takes as input the table D which is a collection of tuples, the ontology graph G, and a parameter d_{max} that is used for the approximate feature extraction procedure GETAPPROXFEATURES. The for-loop at Line 2 iterates over each tuple in the base table. *ConceptNodes*(t) denotes the set of concept nodes associated with tuple t of the base table. More than one column of a base table tuple may be associated with ontologies and the value of a column in the tuple may be associated with more than one concept node in the ontology. Hence a single tuple may be associated with multiple concept nodes in the ontology. The feature vector f_t for tuple t is the union of all the features associated with each concept node in *ConceptNodes*(t). The feature vectors for a given table only needs to be computed once. Although the OFFLINEPHASE procedure is a batch processing procedure, the feature vector computation could also be done once for each tuple, when the tuple is inserted into the table.

Algorithm 1 OFFLINEPHASE(D, G, d_{max})

Input: s set D of tuples from the base table, concept graph G, maximum depth d_{max}
Output: a set F of feature vectors for the tuples in D

```
1: $F \leftarrow \emptyset$ /* init */
2: for all $t \in D$ do
3:     $f_t \leftarrow vector(\infty)$ /* init */
4:     for all $c \in ConceptNodes(t)$ do
5:         GETAPPROXFEATURES$(G, c, 0, d_{max}, f_t)$
6:     $F \leftarrow F \cup \{f_t\}$
```

Algorithm 2 GETAPPROXFEATURES(G, c, d, d_{max}, f)

Input: concept graph G, current concept c, current depth d, maximum depth d_{max}, the current feature vector f
Output: approximate feature f

```
1: if $d = d_{max}$ then
2:     return
3: $d \leftarrow d + 1$
4: for all $p \in PropertySet(c)$ do
5:     $f[p] \leftarrow \min(f[p], d)$
6: for all $child \in Adj(G, c)$ do
7:     GETAPPROXFEATURES$(G, child, d, d_{max}, f)$
```

To find the shortest distance between any node and any concept, the simplest way is to invoke the all pairs shortest distance algorithm. The Floyd-Warshall algorithm [6] compares all possible paths through the graph between each pair of vertices. For a graph with V vertices, it is able to do this with $O(V^3)$ comparisons.

For small ontologies, the cubic running time is acceptable; however, for large ontologies such as the NCI Thesaurus that has 64,000 concept nodes, computing the shortest path for all pairs still takes a prohibitive amount of time. In order for our method to be practical, we propose using a depth-first search based algorithm to compute approximate feature vectors. The approximation is based on the intuition that the features most relevant to a given node are usually the nodes that are relatively nearby. Hence, the algorithm for finding the approximate feature vector for a given node c is to traverse the graph starting from c for a maximum of d_{max} steps and maintain the shortest distance for the nodes thus traversed. Algorithm 2 outlines the recursive GETAPPROXFEATURES procedure that implements this approximation. Prior to the first call to GETAPPROXFEATURES, the feature vector f should be initialized to a vector of infinity, i.e., MAXINT, because each entry $f[n]$ stores the (current) shortest distance of the starting node to the node n. As the procedure traversed the graph starting from the given concept node c, the shortest distance stored in f will be updated via Line 5. *Adj*(G, c) denotes the set of concept nodes adjacent to the given concept c in concept graph G. *PropertySet*(c) denotes the set of properties (terminal non-concept nodes) adjacent to concept c in graph G. Note that a breadth-first search implementation is also possible, but requires more space and bookkeeping. Using such depth-limited traversals to compute approximate feature vectors is significantly more efficient for large ontologies, because each feature vector requires a depth-limited traversal that covers

Algorithm 3 OnlineQueryByExample(F, P, n_{trg})

```
Input: a set $F$ of feature vectors for the tuples in the base table, a set $P$ of positive examples from the user
Output: a set $R$ of query results
1: $N \leftarrow RandomSample(F, n_{trg} - |P|)$
2: label $N$ as negative
3: Initialize training set $X \leftarrow P \cup N$
4: model $M \leftarrow$ svm_learn( $X$ )
5: $R \leftarrow$ svm_classify ( $F, M$ )
6: map the labels in $R$ back to base table tuples
```

an asymptotically constant number of nodes and the number of such traversals is at most the size of the base table. As long as the size of the base table is significantly smaller than $O(V^3)$, GetApproxFeatures will be more efficient.

3.2 Learning Query Semantics

Instead of expressing a query explicitly in SQL or any other query language, the user uses a few examples to characterize the query. It is the responsibility of the system to figure out the query semantics. Machine learning techniques are used here to solve the problem.

Training dataset preparation. When the user forms a query in his mind against the base table D, he provides us with a small dataset T where each record in T has the following form:

$$(\mathbf{X}, \texttt{OID}, \texttt{label})$$

where $\texttt{label} \in \{+1, \text{-}1\}$ indicates whether tuple $(\mathbf{X}, \texttt{OID}) \in D$ satisfies the query or not. Note that the user-supplied examples may or may not be part of the base table D.

In order to train a classifier, the training dataset must have both positive and negative examples. But it might be counter-intuitive to ask the users to provide negative examples. In the case where examples provided by the user are all positive (i.e., $\texttt{label} = +1$), we need to generate some negative examples on our own. To do this, we randomly pick a set of tuples $P_n \subset D$ and label each tuple in P_n as negative ($\texttt{label} = -1$). The training dataset thus consists of P_n and the user-provided, all positive examples. Certainly, there is always the possibility that some tuples in P_n actually satisfy the query. But assuming that the tuples satisfying the query account for a small proportion of tuples in D, the classifier usually can overcome such 'noise' in the data.

The OnlineQueryByExample procedure takes as input this set P of positive examples, the set F of feature vectors associated with the base table, and a parameter for the total number of training examples to use. Algorithm 3 outlines the steps of the OnlineQueryByExample procedure.

The algorithm first constructs a set of negative examples for training by taking a random sample of the base table (Line 1) and labeling the corresponding feature vectors as negative (Line 2). Line 3 constructs the training set as the union of the positive and the negative examples. We then learn a model from the training set and apply this model to classify the feature vectors associated with the base table. Note that Line 6 is required, because the mapping between base table tuples to feature vectors is often many-to-one. The learning and the classification occur in the space of feature vectors; hence, we need to map the labels from the classification operation back to the tuples in the base table.

We use support vector machines (SVMs), which we describe below, as the base learner in our work. We then use SVMs to support active learning, which improves our query by example system by reducing the human cost (Section 3.3).

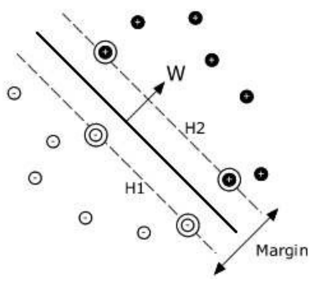


Figure 6: SVM: maximize the margin

Support Vector Machines. Support Vector Machines (SVMs) are a class of supervised learning algorithms introduced by Vapnik [26]. Given a set of training data (positive and negative examples), SVMs learn a linear decision boundary to discriminate between the two classes. The result is a linear classification rule that can be used to classify new test examples. SVMs have demonstrated excellent generalization performance (accuracy on unlabeled data sets) in many domains and have strong theoretical motivation in statistical learning theory [26].

Given a dataset of $(\mathbf{X}_i, \mathbf{Y}_i, \texttt{label}_i)$ where $\mathbf{X}_i$ is the original features, $\mathbf{Y}_i$ is the features derived from the ontology, and $\texttt{label} \in \{+1, -1\}$ is the class label, SVMs learn a decision boundary in the combined feature space of $\mathbf{X}_i$ and $\mathbf{Y}_i$ to separate the two classes. To make our discussion simple, we use $\mathbf{x}_i$ to represent the combined features, that is, $\mathbf{x}_i = \langle \mathbf{X}_i, \mathbf{Y}_i \rangle$, and we assume $\mathbf{x}_i \in R^N$.

An SVM trained from the dataset $(\mathbf{x}_i, \texttt{label}_i)$ specifies a linear classification rule f by a pair $(\mathbf{w}, b)$, where $\mathbf{w} \in R^N$ and $b \in R$, via

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$$

where a point $\mathbf{x}$ is classified as positive if $f(\mathbf{x}) > 0$, or negative if $f(\mathbf{x}) < 0$. Geometrically, the decision boundary is the hyperplane

$$\{\mathbf{x} \in R^N : \mathbf{w} \cdot \mathbf{x} + b = 0\}$$

where $\mathbf{w}$ is a normal vector to the hyperplane and b is the bias.

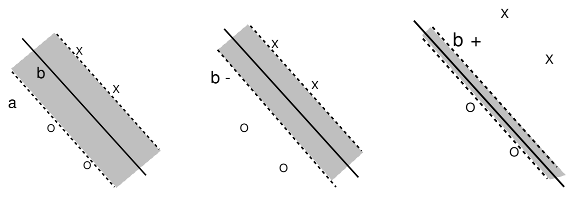
Figure 6 illustrates a decision boundary that separates examples of the positive and negative classes. Clearly, there are many such decision boundaries. The optimal one is the unique boundary that separates positive and negative examples for which the margin is maximized:

$$\max_{\mathbf{w},b} \left\{ min_{\mathbf{x}_i} \{ ||\mathbf{x} - \mathbf{x}_i|| : \mathbf{x} \in R^N, \mathbf{w} \cdot \mathbf{x} + b = 0 \} \right\}$$

The decision boundary shown in Figure 6 is the optimal boundary.

3.3 Active Learning for QBE

We learn a model from the extended dataset of $(\mathbf{X}_i, \mathbf{Y}_i, \texttt{label}_i)(i = 1 \cdots n)$, and use the model to classify the remaining data. One potential challenge is the quality of the labeled examples provided by the user. The user usually provides a very small number of such examples (say 2-5 examples), and it is very likely that classifiers built from such



(a) Current SVM (b) if b is labeled negative (c) if b is labeled positive

Figure 7: Intuition for active learning heuristic. The solid line represents the decision boundary, the shaded region denotes the margin, and 'X' and 'O' denote the positive and the negative support vectors.

small datasets have large variance, which affects the classification accuracy.

In this section, we propose to use active learning to solve this problem. Using the user-provided examples as seeds, we choose a small number of additional examples that are informative in terms of separating the two classes, and ask the user to label the examples.

The size of the training dataset clearly affects the quality of the trained classifier. A larger training dataset reduces model over fitting and improves its accuracy. The goal of active learning is to iteratively select the best example to incorporate into the training dataset so as to obtain a high quality classifier. In our case, we assume that the user has initially provided a small set of labeled examples and our system has computed a set of query results using the algorithms described in Section 3.2. If the user now wants to further refine the query results by labeling, say one more example, our system should use active learning techniques to find the best example for the user to label. For SVMs, several active learning techniques have been proposed in [21, 25]. The problem with many of these techniques is that they require computing the SVM for each possible labeling of a candidate example in order to choose the best example. This is usually impractical in our case, because the number of candidates to consider is the size of the table (usually in the order of thousands) less the labeled examples (usually a handful). Hence, we adapt the simple heuristic approach in [21] for our scenario.

The intuition for the heuristic is illustrated in Figure 7. Suppose the candidate pool of examples consists of points a and b as shown in Figure 7(a). If we pick a and suppose the user labels a as negative, then the SVM and the margin do not change, because the current support vectors are still valid. If we pick b which is closer to the decision boundary and suppose the user labels b as negative (see Figure 7(b), then the margin of the new SVM would have shrunk. Similarly, suppose the user labels b as positive, the margin of the new SVM would also have shrunk. The heuristic proposed in [21] picks the example that is closest to the decision boundary which would ensure that the margin of the SVM gets shrunk by nearly half whether the example is labeled positive or negative.

In order to apply this heuristic, we need compute the distance of each example in the candidate pool to the decision boundary. The distance between any given point $\mathbf{x}$ and the decision boundary of a SVM specified by unit vector $\mathbf{w}$ and

Algorithm 4 GETBESTEXAMPLE(X, F, M)

Input: set of current labeled examples X, set of all examples F, current SVM model M
Output: the best example $f \in F - X$ to present to the user for labeling

```
1: $F' \leftarrow F - X$
2: for all $f \in F'$ do
3:     $d(f) \leftarrow$ calculate the distance from the boundary of $M$ using Equation (1).
4: return $\arg\min_f d(f)$
```

scalar b is given by,

$$d(\mathbf{x}) = |\mathbf{w} \cdot \mathbf{x} - b|. \quad (1)$$

At each iteration of user feedback, our system needs to select the optimal example from the set of unlabeled examples (tuples from the base table) to present to the user for feedback. Algorithm 4 outlines the steps for choosing the best example. The GETBESTEXAMPLE procedure takes as input the set of currently labeled examples, the set of all examples from the base table, and the SVM model obtained from the last iteration. Line 1 finds the set of unlabeled examples. The for-loop at Line 2 iterates over the set of unlabeled examples and compute the their distance to the decision boundary of the current SVM model. The procedure then returns the example that is closest to the boundary. Note that nearest neighbor search techniques do not apply directly to this problem, because the criteria here is searching for points nearest to a given hyperplane. However, quantization techniques can be used on the pool of unlabeled examples to reduce the number of distance computations.

After the best example is selected by the system, it is presented to the user. The user will label the example and the labeled example will be added to the current pool of labeled examples. To form the training set for the SVM learning algorithm, we again may optionally augment the labeled examples with randomly sampled negative examples. The resultant SVM is then used to generate the query result. The user may choose to stop if the results are satisfactory, or request for another iteration of refinement.

4. EXPERIMENTS

We implemented our query by example system using SVM-Light [11] with a linear kernel. For the feature extraction step we used the GETAPPROXFEATURES procedure (Algorithm 2) with the maximum depth set to 7. Our experiments are designed to answer several important questions:

- How does the system perform over different queries?
- How many positive examples does the user need to provide to get reasonable result quality?
- How many negative examples should the system automatically augment?
- How does query selectivity affect result quality?
- How does size of the table affect result quality?
- What kind of response time to expect?

Queries. For the subset of experimental results we present in this paper, we used six queries based on Example 2, varying the anatomic site. Varying the anatomic sites means that each query will be associated with a different subgraph in the ontology with a different number of concept nodes,

Query	Anatomic Site	Nodes	Edges	Properties
Q1	Gastrointestinal System	1063	1503	10264
Q2	Respiratory System	752	952	6841
Q3	Urinary System	401	528	3765
Q4	Hematopoietic and Lymphatic System	1142	1831	12174
Q5	Reproductive System	1176	1614	10833
Q6	Head and Neck	1080	1584	11279

Table 3: Queries used in the experiments.

concept edges, and concept properties in the subgraph. The characteristics of the six queries are summarized in Table 3.

Dataset. We used a real ontology in our experiments, namely, the NCI Thesaurus [17] and generated synthetic base table data using values from the NCI Thesaurus. The NCI Thesaurus is a DAG with 63,924 concept nodes, 72,466 concept edges, and 540,298 properties. The schema of the base table follows that of Figure 1. Given a particular query, a table size and a target query selectivity, we generate the base table as follows. Use the query to generate the two sets of diagnosis terms: the positive set contains all distinct possible terms that satisfy the query, and the negative set contains all distinct terms that do not satisfy the query. Pick randomly with replacement from the positive set enough number of terms to fulfill the target query selectivity. Pick randomly with replacement from the negative set to fill up the rest of the table. For each table and query we also store the *ground truth*, i.e., which tuples satisfy the query, which do not.

In order to simulate a real user supplying only a few positive examples, we generate the training data for a given table and query as follows. Using the ground truth for that query and table, extract the positive tuples from the table. Pick randomly without replacement the required number of positive examples from the positive tuples. To make up the required number of training examples, pick the required number of examples from the table and assume that they are negative examples. We will show in one of our experiments the effect of this assumption.

Performance Metrics. In addition to processing time which consists of time to learn the model, and time to classify the tuples in the table using the model, we measure the result quality returned by our query by example system. Result quality is measured using three metrics. *Accuracy* is the percentage of tuples in the table classified correctly. *False positive rate* is the percentage of tuples in the table classified as positive even though they do not satisfy the query. *False negative rate* is the percentage of tuples in the table classified as negative even though they do satisfy the query. The measurement of these metrics are averaged over 10 random runs. Each run uses a different randomly generated training dataset.

4.1 Varying the number of positive examples

In this experiment, we investigate how the number of positive examples supplied by the user affects the result quality. Figure 8 shows the result quality measurements for all six queries as the number of positive examples vary from 2 to 16. The size of the table is fixed at 10,000, the query selectivity[1] at 10%, the size of the training data at 40 examples. Note that we purposely chose rather small numbers of positive examples, because it is not practical to expect a human user to supply tens or hundreds of examples. Figure 8(a) shows that accuracy generally improves with more positive examples. Even with two positive examples, our system is able to achieve 90% accuracy. Figure 8(b) shows that the false positive rate is generally quite stable and very close to zero (note the scale on the y-axis). Figure 8(c) shows that the false negative rate decreases significantly with more positive examples thus contributing to the increase in accuracy. This result is expected, since the SVM would be able to learn a better model for the positive examples if there are more of them and would then be able to more correctly classify a positive example as positive. Note also that the empirical results are consistent over all six queries.

[1]Query selectivity is the size of the query result as a percentage of the size of the table.

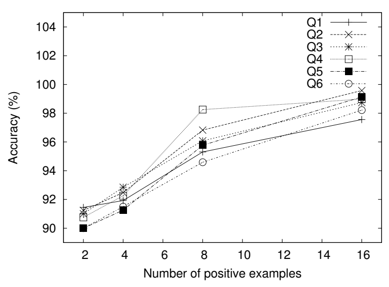
4.2 Varying the number of training examples

In this experiment, we study how the number of training examples and the number of randomly selected negative examples affect the query result quality. We fixed the number of positive examples at 8, and vary the total number of training examples from 40 to 160. The size of the table and the query selectivity remains at 10,000 and 10% respectively. Figure 9 shows the results of our study for a representative query Q2. Figure 9(a) shows that accuracy drops by about 5% as the number of training examples are increased. This result seems counter-intuitive, since we expect higher accuracy with more examples even if they are negative examples. However, recall that since we do not expect the user to supply any negative examples, our system automatically generate negative examples by random sampling from the table and assigning negative labels no matter what the ground truth might be. Hence the randomly generated negative examples might contain labeling errors.

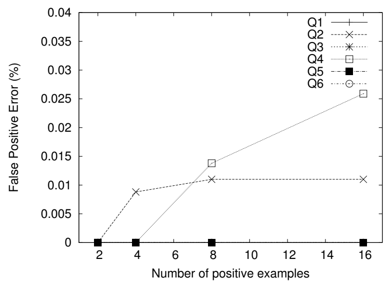
To corroborate this reasoning, we re-generated the training data by picking the negative examples not randomly from the table, but from the true negative set of examples computed using the ground truth. Figure 9(b) shows the result quality using these perfect training data. With the perfect training data, the result quality remains stable as the number of training examples are increased which suggests that the increase in errors in Figure 9(b) are due to the randomly generated negative examples introduced by our system. Since these negative examples are generated via random sampling, the number of mislabeled negative examples should be proportional to the query selectivity for that table and query. Figure 10 plots the accuracy and the false negative rate against number of training examples for 4 different query selectivity. The plots clearly show that when the query selectivity is high, increasing the number of training examples (and hence the number of randomly generated negative examples) will increase the false negative rate. The false positive rates remain very low and the corresponding plot is omitted for brevity.

4.3 Varying the query selectivity

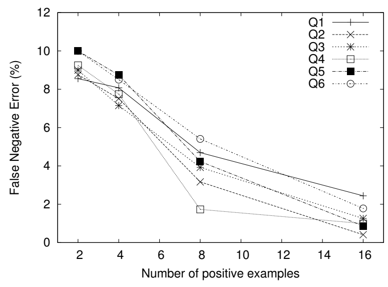
The previous experiment motivates the need to understand the relationship between query result quality and the query selectivity. Figure 11 plots query result quality against query selectivity where the table size is held constant at 10,000, the number of training examples at 160, and the number of positive examples at 8. The plot suggests that the relationship between query selectivity and result quality



(a) Accuracy

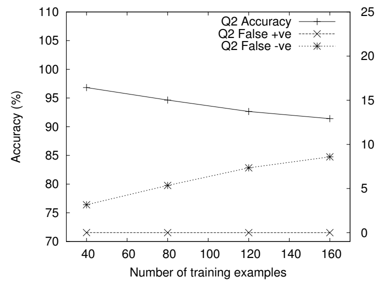


(b) False Positive Rate

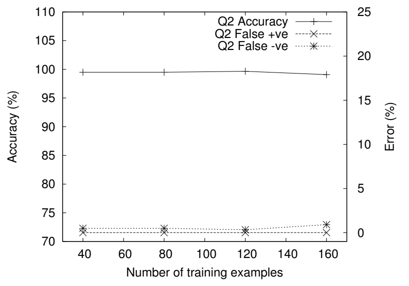


(c) False Negative Rate

Figure 8: Query result quality over 6 queries and varying numbers of positive examples.

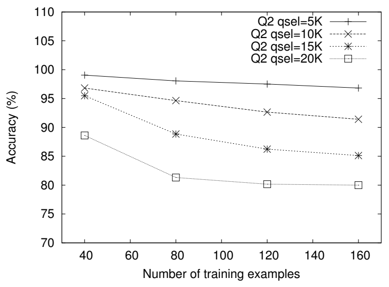


(a) Our method: random negative examples

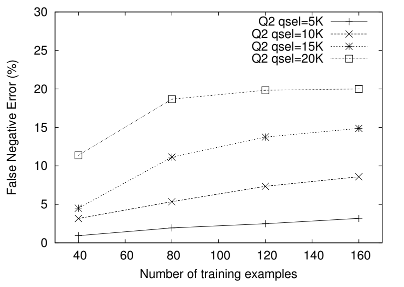


(b) Perfect Training Data

Figure 9: Query result quality for Q2 over varying training data size. Accuracy is plotted against the left axis, and false positive and negative error rates are plotted against the error axes on the right.



(a) Accuracy across query selectivity



(b) False negative rate across query selectivity

Figure 10: Query result for Q2 over varying training data size and 4 different query selectivity values.

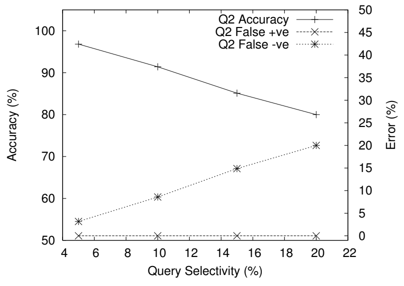


Figure 11: Result quality over different query selectivity.

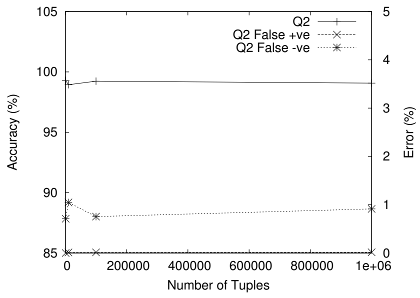
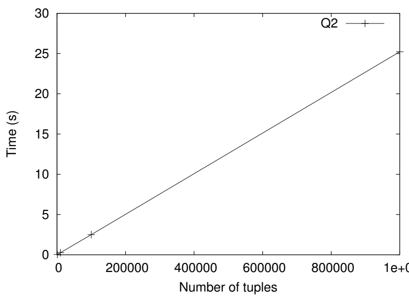
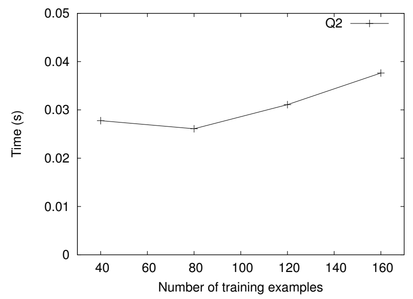


Figure 12: Result quality over table size.



(a) Classification Time



(b) Learning Time

Figure 13: Processing Time.

Area of Expertise	Number of users
Medical	2
Database	4
Ontology	2
Database + Ontology	1

Table 4: Background of the nine users in our study.

is only linear.

4.4 Varying the table size

In this experiment we investigate the effect of table size on the query result quality. Figure 12 shows our query result quality measurements for different table sizes, holding the query selectivity constant at 5%, the number of training examples at 160, the number of positive examples at 16. It is somewhat surprising to observe that the size of the table does not have any significant effect on the result quality. The small training data set is enough to completely capture the query semantics even when the underlying set of tuples is large.

4.5 Processing time

We require the query result to be produced in a reasonable amount of time. In this experiment, we investigate the response time characteristics of our system. We measured the time for the system to learn a model from the training data and the time for the classifier to classify all the tuples in the base table. Figure 13 shows the learning time and the classification time for query Q2 on an Intel Xeon 3.2 GHz machine running Linux. Note that the running time includes reading and writing the base data and the result file from and to disk. Even so, the learning time is negligible, and the classification time is linear in the size of the base table. Processing a query by example on a table of size 10,000 takes only a quarter of a second (including disk access). Note the current prototype has not incorporated several possible optimizations, such as performing classification only on the distinct feature vectors from the base tables, and hence we expect even better processing times in the optimized version.

5. USER STUDY

In order to demonstrate the usability of the concept of semantic query by example using our proposed method, we conducted a user study on nine users with different levels of expertise in terms of information technology, medical knowledge, and knowledge representation. Table 4 summarizes the background of our subjects. Our user study is based on the EMR scenario introduced in Section 1. The goals of our study are to validate if our proposed method is indeed user friendly and to evaluate the accuracy of our method when it is used by different users.

User tasks. Each user is asked to perform three tasks. Each task is associated with a particular information need. In our study we used the first three queries in Table 3 as the information need for the three tasks.

For each task, the user is given an explanation of the EMR database and the information need associated with the task (eg., find all patients with some disease in the gastrointestinal system), and then asked to perform two steps. In the first step, the user is asked to write down her best effort SQL query to retrieve the required records from the database. In the second step, the user is asked to identify approximately 10-15 records that satisfy the information need from a pool of about 40 randomly picked records from the EMR database. For medical experts, we also asked them to provide diagnosis terms that are associated with the information need.

Methodology. We interviewed the users about their experience in order to get a qualitative answer to the usability of the two steps, and we timed the users for each task and each step in order to quantify the "ease of use" of the two steps. After we obtained the positive examples from each user (for step 2 of each task), we ran our SVM-based algorithm to obtain the required results for each task (i.e. query) and computed the accuracy, false positive and false negative rates. These measurements are averaged over all the user subjects and are tabulated in Table 5 along with their standard deviation. The measurements associated with each of the three task are collated in one row. For each task, we present the accuracy of our approach when using the 15 user-marked positive examples from each of the nine users as input. The accuracy is measured by false positive rate and false negative rate. We compute the mean and standard deviation for each accuracy measurement. We also present the time (in minute) it took for the users to write down the best-effort SQL query and to mark the positive examples. We compute the mean and standard deviation for the each time measurement.

Observations. Eight out of the nine users reported difficulty in writing down the SQL query and dissatisfaction with the query they finally wrote since they intuitively knew that their query is not the "right" query. One user (who is an ontology and database expert) chose to write a two-page program for each task. It took about four minutes on average for a user to write down her best-effort query for each task. All nine users reported a pleasant experience marking the qualified tuples. It took about two minutes in average for a user to mark the 15 qualified tuples for each task. It took a medical expert about three minutes to came up with a small set of her own positive examples for each task. We tried to run the SQL queries that the user subjects wrote, but unfortunately all of them do not run without significant corrections (hence we are not able to quantify the accuracy of these queries rigorously). Even when we ran the corrected versions of these queries, we found that the accuracy of our learning-based approach is still much higher than that of the best-effort SQL query written by any user during the study regardless of the fact that some of the users are database experts. In summary, our user study demonstrates that identifying positive examples from a pool is more user friendly than writing SQL queries (even very simple keyword-based filtering queries) and that the quality of the results computed by our semantic query by example method is quite reasonable and acceptable in most non-critical applications.

6. CONCLUSION

In this paper, we introduce a machine learning approach to support semantic queries in relational database. In semantic query processing, the biggest hurdle is to represent ontological data in relational form so that the relational database engine can manipulate the ontology in a way consistent with manipulating the data. Previous approaches include transforming the graph ontological data into tabular form, or representing ontological data in XML and lever-

	Accuracy		False Positive		False Negative		Time (min) to Write SQL		Time (min) to Mark Examples	
Task	Mean	Std.dev.	Mean	Std.dev.	Mean	Std.dev.	Mean	Std.dev.	Mean	Std.dev.
1	89.9	0.028	0.098	0.028	10.000	0	5.25	2.31	2.00	0.93
2	89.9	0.011	0.085	0.011	10.001	0	4.13	2.10	2.25	0.76
3	89.8	0.033	0.176	0.036	10.000	0	3.63	2.45	2.06	1.27

Table 5: Accuracy rate, false positive rate, false negative rate and timing measurements for our user study.

aging database extenders on XML such as DB2's Viper. These approaches, however, are either expensive (materializing a transitive relationship represented by a graph may increase the data size exponentially) or requiring changes in the database engine and new extensions to SQL. Our approach shields the user from the necessity of dealing with the ontology directly. Indeed, as our user study indicates, the difficulty of expressing ontology-based query semantics in a query language is the major hurdle of promoting semantic query processing. With our approach, the users do not even need to know ontology representation. All that is required is that the user gives some examples that satisfy the query he has in mind. The system then automatically finds the answer to the query. In this process, semantics, which is a concept usually hard to express, remains as a concept in the mind of user, without having to be expressed explicitly in a query language. Our experiments and user study results show that the approach is efficient, effective, and general in supporting semantic queries in terms of both accuracy and usability.

7. REFERENCES

[1] A. Balter. *Mastering Microsoft Office Access 2007 Development*. Sams, 2007.

[2] M. Belkhatir, P. Mulhem, and Y. Chiaramella. A conceptual image retrieval architecture combining keyword-based querying with transparent and penetrable query-by-example. In *ACM International Conference on Image and Video Retrieval (CIVR)*, pages 528–539, 2005.

[3] G. Boccignone, A. Chianese, V. Moscato, and A. Picariello. Animate system for query by example in image databases. In *EuroIMSA*, pages 451–456, 2005.

[4] A. K. Choupo, L. Berti-Equille, and A. Morin. Optimizing progressive query-by-example over pre-clustered large image databases. In V. Benzaken, editor, *BDA*, 2005.

[5] ASTM E2369-05 Standard Specification for Continuity of Care Record (CCR). `http://www.astm.org`.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, second edition, 2001.

[7] S. Das, E. I. Chong, G. Eadon, and J. Srinivasan. Supporting ontology-based semantic matching in RDBMS. In *VLDB*, pages 1054–1065, 2004.

[8] Health level seven. `http://www.hl7.org`.

[9] International Classification of Diseases, Ninth Revision, Clinical Modification (ICD-9-CM). `http://www.cdc.gov/nchs/about/otheract/icd9/abticd9.htm`.

[10] Systematized Nomenclature of Medicine-Clinical Terms. `http://www.ihtsdo.org/`.

[11] T. Joachims. SVM-Light support vector machine. `http://svmlight.joachims.org/`.

[12] R. Krishnamurthy, S. P. Morgan, and M. M. Zloof. Query-by-example: Operations on piecewise continuous data (extended abstract). In *VLDB*, pages 305–308, 1983.

[13] L. Lim, H. Wang, and M. Wang. Semantic data management: Towards querying data with their meaning. In *ICDE*, pages 1438–1442, 2007.

[14] L. Lim, H. Wang, and M. Wang. Unifying data and domain knowledge using virtual views. In *VLDB*, pages 255–266, 2007.

[15] L. Lim, H. Wang, and M. Wang. Semantic queries in databases: Problems and challenges. *CIKM*, 2009.

[16] L. Ma, Z. Su, Y. Pan, L. Zhang, and T. Liu. RStar: An RDF storage and query system for enterprise resource management. In *CIKM*, 2004.

[17] National cancer institute thesaurus. `http://www.nci.nih.gov/cancerinfo/terminologyresources`.

[18] OntoBroker. `http://ontobroker.aifb.uni-karlsruhe.de/index_ob.html`.

[19] OTK tool repository: Ontoedit. `http://www.ontoknowledge.org/tools/ontoedit.shtml`.

[20] N. Rasiwasia, N. Vasconcelos, and P. J. Moreno. Query by semantic example. In *ACM International Conference on Image and Video Retrieval (CIVR)*, pages 51–60, 2006.

[21] G. Schohn and D. Cohn. Less is more: Active learning with support vector machines. In *ICML*, pages 839–846. Morgan Kaufmann, 2000.

[22] SWAD-Europe Deliverable 10.2: Mapping Semantic Web Data with RDBMSes. `http://www.w3.org/2001/sw/Europe/reports/scalable_rdbms_mapping_report/`.

[23] The KArlsruhe ONtology and semantic web tool suite. `http://kaon.semanticweb.org/`.

[24] The protégé ontology editor and knowledge acquisition system. `http://protege.stanford.edu/`.

[25] S. Tong and D. Koller. Support vector machine active learning with applications to text classification. In P. Langley, editor, *ICML*, pages 999–1006. Morgan Kaufmann, 2000.

[26] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer Verlag, New York, 1995.

[27] M. M. Zloof. Query-by-example: the invocation and definition of tables and forms. In D. S. Kerr, editor, *VLDB*, pages 1–24. ACM, 1975.

[28] M. M. Zloof. Query-by-example: A data base language. *IBM Systems Journal*, 16(4):324–343, 1977.