Efficient Inference Query Processing for Large-scale T-Box

Insung Kim and Jonghoon Chun
Department of Computer Engineering, Myongji University
Nam-Dong 38-2, Cheoin-Gu, Yongin, Gyeonggi-Do 449-728, Korea
{iskim, jchun}@mju.ac.kr

Queries involving inference are known to be difficult to process. When dealing with large-scale T-Box, the problem becomes even more complex. In this paper, we propose a new method to efficiently process inference queries for large-scale T-Box. We use a simple bit matrix to summarize the relationships between classes, and use it as an index to speed up the query processing. We show that our method outperforms others and the performance gap becomes larger as the size of ontology grows.

Key Words: Inference query processing, ontology, T-Box, OWL, relational databases, bit matrix, class, property

1. INTRODUCTION

With the emergence of Semantic Web, the next generation computer can understand the meaning as well as infer new facts based on others previously defined. Such capability became widely available with development of practical ontologies. In order to manage large-scale ontologies, proven relational database technologies are often employed not only to store them but also to efficiently process queries involving inferences[2-8]. However, most of researches confine themselves focused on manipulation of A-Box instances leaving interesting research topics wide open on T-Boxes. Large-scale T-Boxes are not that difficult to find around as UMLS[15] is probably the best well-known large-scale T-Box example with 250,000 axioms stored and managed. It is not too difficult to imagine how hard it would be to process queries involving multiple steps of inferencing within such ontologies.

In this paper, we propose a new mechanism to efficiently process inference queries with the help of an index. We present how classes, properties, and relationships between them are converted into a relatively simple bit matrix and devise algorithms to utilize the bit matrix as an index to process queries which might require costly inferences. In section 2, we investigate related works mainly on ontology handling with relational database management systems. Section 3 defines the inference based on each characteristics of OWL properties. Section 4 introduces bit matrix representation of T-Box, and proposes algorithms for efficient inferences using the bit matrix as the index. Section 5 presents experimental results. Section 6 is the conclusion.

2. RELATED WORKS

Ontologies in general, are composed of two parts namely T-Box and A-Box[10]. T-Box defines concepts as classes, properties and relationships between them, while A-Box defines instances on the top of concepts defined by T-Box. Jena[2],
Sesame\cite{5} and DLDB\cite{7} all proposed to use relational database management systems for storage and manipulation of large-scale ontology. Jena proposed vertical schema based on relational model, and stored ontology in the form of triples. However, in order for Jena to process inference queries, large number of self-joins are necessary, which will result in high cost. Sesame\cite{5} uses binary schema to model classes and properties separately and stored in the form of binary tables. Thus it avoids redundant storage of subjects and predicates. DLDB\cite{7} uses hybrid schema that mixes vertical schema and binary schema. The hybrid schema represents classes and object properties as tables, and data type properties as attributes. However, all of the above do not support efficient inference mechanisms taking into account OWL’s property characteristics. ONTOMS\cite{14} uses a relational database management system while proposing inference mechanism based on OWL’s property characteristics. However, they all focused their research on inferencing within A-Box only leaving inference on large-scale T-Box remained as an interesting yet untouched research topic. If we consider processing inference queries in large-scale ontologies such as UMLS\cite{15} which has 200,000 concepts and 50,000 roles, any of the above work would be very inefficient. We propose to use index in the form of bit matrix in order to support efficient inference query processing. Our inferencing mechanisms are based OWL’s property characteristics.

3. INFEERENCE BASED ON OWL’S PROPERTY CHARACTERISTICS

Properties in OWL may have five characteristics\cite{16}. Those are inverse of, transitive, symmetric, functional, and inverse functional. If property $P$ and $P'$ are inverse of each other, then $P(x, y) \Rightarrow P'(y, x)$. If property $P$ is transitive, then $P(x, y) \land P(y, z) \Rightarrow P(x, z)$. If property $P$ is symmetric, then $P(x, y) = P(y, x)$ and implies $P$ is transitive. If property $P$ has functional characteristic, then $P(x, y) \land P(x, z) \Rightarrow y$ equals to $z$. If property $P$ has inverse functional characteristic, then $P(y, z) \land P(z, x) \Rightarrow y$ equals to $z$.

In this paper, we will not cover inference mechanisms involving functional and inverse functional characteristics, since those two are only used to inference new facts related to instances within the same class. Therefore we only consider three properties that are necessary for T-Box inference –inverse of, transitive, and symmetric.

When a new concept is added to a T-Box, inference performs as follows according to each property characteristics.

**Definition 1** (Add Transitive property). When $P(C_i, C_j)$ is added to the T-Box and if $P$ is transitive; infer the relationships $P(C_x, C_y)$ for those $C_x$ and $C_y$ that satisfy $P(C_x, C_i)$ and $P(C_j, C_y)$, where $P$’s are properties and $C_i$’s are classes.

**Definition 2** (Add symmetric property). When $P(C_i, C_j)$ is added to the T-Box and if $P$ is symmetric; first infer the relationship $P(C_i, C_j)$, second infer the relationships $P(C_o, C_j)$ and $P(C_o, C_i)$ for those $C_o$ and $C_i$ that satisfy $P(C_o, C_j)$ and $P(C_o, C_i)$ respectively, where $P$’s are properties and $C_i$’s are classes.

**Definition 3** (Add Inverse of property). When $inverseOf(P, P')$ is added to the T-Box; infer the relationships $P'(C_b, C_a)$ and $P(C_b, C_d)$ for those $C_a$, $C_b$, $C_c$, and $C_d$ that satisfy $P(C_a, C_b)$ and $P'(C_c, C_d)$ respectively, where $P$’s are properties and $C_i$’s are classes.
4. IMPLEMENTATION

4.1 Index Representation for T-Box

We propose to represent T-Box in terms of a bit matrix, in other words classes, properties and relationships between them are represented by bits. When there are \( n \) properties, we assign integers between 0 and \( n-1 \) as unique identifiers to distinctly indentify each property. Each property is represented as a bit vector (which is a simple array of bits) and for \( n \) properties we need to create \( n \) bit vectors. Value of the property identifier is used as the position to set the corresponding bit to one in each bit vector. The number of bit vectors as well as the length of each bit vector equals to \( n \). For example, in Table 1 five property bit vectors are shown.

When there are \( m \) classes, we assign integers between 0 and \( m-1 \) as unique identifiers to distinctly identify each class. To represent relationships between classes, we make an \( m \times m \) matrix. In each cell of the matrix we store one bit vector, and that bit vector may be a combination of one or more bit vectors. The resultant bit vector that gets stored is computed by ORing number of bit vectors corresponding to the same \( i_{th} \) row and \( j_{th} \) column of the matrix. In a bit matrix, \( P_x(C_i, C_j) \) is represented by a bit set to one in \( P_x \)'s position of the bit vector in \( i_{th} \) row and \( j_{th} \) column. Table 2 shows the identifiers for the classes and Table 3 shows T-Box concepts. Table 4 shows resultant bit matrix on concepts in Table 3. We use bit matrix as an index to speed up the inference query processing.

<table>
<thead>
<tr>
<th>Name</th>
<th>Identifier</th>
<th>Bit vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISA</td>
<td>0</td>
<td>10000</td>
</tr>
<tr>
<td>hasMember</td>
<td>1</td>
<td>01000</td>
</tr>
<tr>
<td>has Advisor</td>
<td>2</td>
<td>00100</td>
</tr>
<tr>
<td>degreeFrom</td>
<td>3</td>
<td>00010</td>
</tr>
<tr>
<td>has Alumnus</td>
<td>4</td>
<td>00001</td>
</tr>
</tbody>
</table>

Table 1. Example Property Bit Vectors

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person</td>
<td>0</td>
</tr>
<tr>
<td>Professor</td>
<td>1</td>
</tr>
<tr>
<td>Student</td>
<td>2</td>
</tr>
<tr>
<td>University</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 2. Identifier of classes

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Professor</td>
<td>ISA</td>
<td>Person</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Student</td>
<td>ISA</td>
<td>Person</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Student</td>
<td>has Advisor</td>
<td>Professor</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>University</td>
<td>has Alumnus</td>
<td>Student</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3. Defined concepts of T-Box

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Professor</td>
<td>ISA</td>
<td>Person</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Student</td>
<td>ISA</td>
<td>Person</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Student</td>
<td>has Advisor</td>
<td>Professor</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>University</td>
<td>has Alumnus</td>
<td>Student</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4. Representation of bit matrix

4.2 Inference Algorithms

When a query searching for \( C_x \) satisfying \( P(C_i, C_x) \) is submitted, we use the bit matrix as an index to retrieve \( i_{th} \) row and gather all the bit vectors in that row and use \( P \)'s location to subsequently find related \( C_x \)'s. In a similar way, in order to find \( C_x \) satisfying \( P(C_x, C_i) \), we start from \( i_{th} \) column instead and the rest is similar to the above. Each is described in Algorithm 1 and 2.
Algorithm 1. Find classes from row data

Input : class C, property P
Output : class list L

L ← NULL
for (i = 0; i < bitmatrix.columnCount; i++) {
    if bitmatrix[C][i] ∧ P is true then
        L ← i-th class
    end if
}

Algorithm 2. Find classes from column data

Input : class C, property P
Output : class list L

L ← NULL
for (i = 0; i < bitmatrix.columnCount; i++) {
    if bitmatrix[i][C] ∧ P is true then
        L ← i-th class
    end if
}

When \( P(C_i, C_j) \) is added to the T-Box and if \( P \) is transitive; find \( C_x \) and \( C_y \) from the bit matrix satisfying \( P(C_x, C_i) \) and \( P(C_j, C_y) \) using Algorithm 1 and 2, and add \( P \)'s bit vector to the cell \((x_{th}, y_{th})\) as shown in Algorithm 3.

Algorithm 3. Transitive property inference

Input : subject class Cs, object class Co, property P
Output : Inference Bit Matrix

subject class list S ← NULL
object class list O ← NULL
S ← Cs + FindClassesFromColumn(Cs, P)
O ← Co + FindClassesFromRow(Co, P)
for (i = 0; i < S.length; i++) {
    for (j = 0; j < O.length; j++) {
        Add Inference Bit Matrix \((S_i, P, O_j)\)
    }
}

When \( P(C_i, C_j) \) is added to the T-Box and if \( P \) is symmetric; first, add \( P \)'s bit vector to the cell position \((j_{th}, i_{th})\). Second, find \( C_x \) and \( C_y \) from the bit matrix satisfying \( P(C_i, C_x) \) and \( P(C_j, C_y) \) using Algorithm 1 and 2 and add \( P \)'s bit vector to the cell \((x_{th}, y_{th})\).

Algorithm 4. Symmetric property inference

Input : subject class Cs, object class Co, property P
Output : Inference Bit Matrix

Add Inference Bit Matrix \((Co, P, Cs)\)

subject class list S ← NULL
object class list O ← NULL
S ← Cs + FindClassesFromRow(Cs, P)
O ← Co + FindClassesFromRow(Co, P)
for (i = 0; i < S.length; i++) {
    for (j = 0; j < O.length; j++) {
        Add Inference Bit Matrix \((S_i, P, O_j)\)
    }

Proceedings of the Third International Conference on Emerging Databases (EDB 2011)
Add Inference Bit Matrix \{(O_i, P, S_i)\} 

Algorithm 4. Symmetric property inference

When \text{InverseOf}(P_i, P_j) is added to the T-Box; find \(C_a, C_b, C_c\) and \(C_d\) satisfying \(P_i(C_a, C_b)\) and \(P_j(C_c, C_d)\), add \(P_j\)'s bit vector in the cell \((b_{th}, a_{th})\), and add \(P_i\)'s bit vector to the cell \((d_{th}, c_{th})\).

Algorithm 5. AddInverseOfProperty

Input : property \(P\), inverse property \(P'\)
Output : Inference Bit Matrix

\[
\begin{align*}
\text{Subject, Object Class set} \{C_{s1}, C_{o1}\} & \leftarrow \text{FindClassesSet}(P) \\
\text{Subject, Object Class set} \{C_{s2}, C_{o2}\} & \leftarrow \text{FindClassesSet}(P') \\
\text{for} (i=0; i<\{C_{s1}, C_{o1}\}.length; i++) { & \text{Add Inference Bit Matrix} \{C_{o1}, P', C_{s1}\}; } \\
\text{for} (i=0; i<\{C_{s2}, C_{o2}\}.length; i++) { & \text{Add Inference Bit Matrix} \{C_{o2}, P, C_{s2}\}; } \\
\end{align*}
\]

Algorithm 6. Inverse of property inference

Algorithm 6 is the main part of the query processing. In the first phase, we find classes and instances that are directly connected to the given query. In the second phase, we look up the bit matrix using the classes obtained in the first phase to get the result inferred by the T-Box. The result of second phase is in the form of triples. The resultant set of triples drastically reduces the scope of our search, therefore we obtain huge benefit in terms of search performance.

Algorithm 6. FindInferenceInstances

Input : SearchKey value
Output : ResultInstance values

\[
\begin{align*}
\text{Classes and id List Set} L<\text{C, id}> & \leftarrow \text{NULL} \\
\text{Property List} P & \leftarrow \text{NULL} \\
\text{Value List} V & \leftarrow \text{NULL} \\
L<\text{C, id}> & \leftarrow \text{find classes and instances in inverted index(value)} \\
\text{for} (i = 0; i < L.length; i++) { & \text{Pi} \leftarrow \text{find property in Bit Matrix} \{Li\} \\
V & \leftarrow V + \text{select values from} Li<\text{C}>, \text{Pi} \text{ where id=Li<id>} } \\
\text{return} V \\
\end{align*}
\]

Algorithm 6. Inference query processing

Proceedings of the Third International Conference on Emerging Databases (EDB 2011)
5. EXPERIMENTAL EVALUATION

We compared time needed to complete inference query processing with our proposed method to iS, KAON2 and Racer. In [16], the experimental results were obtained by large-scale A-Box instead T-Box. So we tried our best to simulate the same experimental environment as in [16]. We set the number of triples of our T-Box equal to the number of triples used, and tried to set up similar hardware environments. The inference query processing time in our method include time needed to look up the necessary information in the bit matrix and to finally get the instances and retrieve them.

As shown in Figure 1, our approach outperforms all others by far. In the experiment we did not include the time needed to initially load ontology as iS, KAON2 and Racer all need extra time for initial loading. Similarly our method also need additional time to build the bit map matrix, and also did not include that time in the evaluation.

![Figure 1. Comparison to iS, KAON2, and Racer](image)

6. CONCLUSION

In this paper, we proposed a new method for efficiently process inference queries. We used a bit matrix to simply represent large-scale T-Box and used it as the index to speed up the query processing. We showed that our method outperforms others by far. Future works include mostly on performance issues such as the compression of bit matrix and the parallel index search. We believe that there are still lots of room for improvement in our method.

REFERENCES


