Query Translation Scheme for Heterogeneous XML Data Sources

Cindy X. Chen[1], George A. Mihaila[2], Sriram Padmanabhan[3], Isabelle M. Rouvellou[2]

[1]Department of Computer Science
University of Massachusetts at Lowell
Lowell, MA 01854, U.S.A.
cchen@cs.uml.edu

[2]IBM T. J. Watson Research Center
P. O. Box 704
Yorktown Heights, NY 10598, U.S.A.
mihaila,rouvellou@us.ibm.com

[3]IBM Santa Teresa Lab
555 Bailey Ave.
San Jose, CA 95141, U.S.A.
srp@us.ibm.com

ABSTRACT

In order to formulate a meaningful XML query, a user must have some knowledge of the schema of the XML documents to be queried. The query will succeed only if the schema of the actual documents is consistent with the user's information. When a user queries a collection of documents collected from heterogeneous XML data sources, there is a high possibility that these documents do not all conform to the same schema assumed by the user, thus causing the query to fail. In this paper, we try to solve this *query and data schema mismatching* problem by proposing a query translation scheme. Without attempting to solve the general problem of schema integration, we present an *inclusion mapping* algorithm that decides how compatible the schema of the query and the schema of the target XML documents are. Based upon the compatibility, the query will be executed directly, or translated according to the target schema and then executed, or rejected.

Categories and Subject Descriptors

H.2.5 [**Heterogeneous Databases**]: Program translation

General Terms

Management

Keywords

XML, Heterogeneous Databases

1. INTRODUCTION

Traditional database management systems manipulate collections of strongly structured data: all tuples in a certain relation have the same attributes, with fixed types. These systems expect data to conform to a regular structure, and enforce strong typing in queries, therefore are able to perform certain optimization when accessing the data.

In contrast, with the Web growing as the biggest database ever existed, the problem of how to access data from distributed and heterogeneous sources has generated a significant interest in the database research community for scalable database integration architectures.

Most of the data on the Web is *semistructured* data, which has no absolute schema fixed in advance, and the data structure may be irregular or incomplete. In practice, even if all the data on the Web are *semantically compatible*, chances are that the data is represented using different schemas. In order to access data from different sources, users need to resolve the differences among schemas. Thus, schema integration and reconciliation has become a well-studied problem and various solutions have been proposed in the literature [1, 2, 4, 5, 6, 7, 9]. For example, [4] combines matching predictions from a set of machine learning techniques based on element name matching, content matching, text classification and domain-specific knowledge. The Cupid system [6] uses a tree similarity metric based on both linguistic and structural information. The system presented in [2] uses approximate concept translation to mediate between sources using different representations of related concepts. However, the complexity and variety of real-life schemas continue to make automatic schema integration a hard problem in practice. A flexible technique presented in [5] for mapping a query to a schema allows users to match a schema that has the required elements in a different order than that specified in the query.

However, we have observed that when users searching the Web, they may not know the information about the schema of the target data well; and they do not require the schemas of data from different sources to be the same. So, in this paper, without attempting to solve the general problem of schema integration, we propose a solution to search data gathered from heterogeneous data sources—the Web, for example.

Therefore, we focus on the problem of formulating meaningful XML queries across various data sources. We present an algorithm that decides whether an XML query schema is sufficiently compatible with the target data schema by generating an *inclusion mapping* with a *compatibility factor*. If the schemas are not identical but compatible, the generated mapping is then used to translate the query according to the

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
WIDM'05, November 5, 2005, Bremen, Germany.
Copyright 2005 ACM 1-59593-194-5/05/0011 ...$5.00.

target XML data schema. The query and data schemas are considered compatible if the *compatibility factor* of a mapping is above or equal to a user specified or system default threshold.

In order to estimate the compatibility between queries and data schemas, we introduce a *tree similarity* measure that takes into account both the *semantic similarity* between element names and the *structural compatibility* of the two schema structures. Our matching algorithm takes advantage of the hierarchical nature of the XML data model by limiting the search space in order to improve performance. Thus, instead of performing an exhaustive search for matching elements in the schema, we explore only those possibilities that can improve the overall matching accuracy. We also show that the running time is affected more by the complexity of the query than the size of the schema. Thus, for an important class of practical queries, the algorithm performs well even for large schemas.

The paper is organized as follows: the next section explains our motivation; Section 3 presents the *inclusion mapping* algorithm; Section 4 describes how we implement the system and shows experiment results; and Section 5 concludes the paper.

2. PROBLEM DEFINITION

In traditional databases, when a user issues a query, he/she usually knows the schema of the database to be queried. If the query schema is different from the database schema, the query is rejected. However, when dealing with XML data, especially when it originates from the Internet, users may not know exactly what the schema is. Furthermore, similar XML data from different sources may have different schemas. As a result, users may need to write queries without complete knowledge of the schema of the data to be queried; or they may write one set of queries against one known source and later want to apply it to data from other sources. In either case, a query should not be simply rejected if its schema is not *identical with* the schema of the data. Instead, there should be a query translation scheme that translate the query into some form compatible to the data schema.

Following are two practical examples of this problem. Example 1 is based on a B2C scenario, where a user wants to find out where an item is delivered. The schema shown in Figure 1 is part of the purchase order schema from [10].

```
<xsd:element name="purchaseOrder"...>
  <xsd:element name="shipTo"...>
  <xsd:element name="billTo"...>
  <xsd:element name="items"...>
       ......
```

Figure 1: Sample XML Schema

Example 1. *Find where an article is delivered.*

```
FOR $p IN //purchaseOrder
RETURN
      $p/deliverTo
```

In the schema, there is no element named "*deliverTo*" but there is an element named "*shipTo*". Though these two terms are different, they may contain similar information that the user wants. Simply rejecting the query will not be the optimal choice in this case. Instead, the system could automatically *infer* a mapping and translate the query to:

```
FOR $p IN //purchaseOrder
RETURN
      $p/shipTo
```

This new query will then be evaluated on the XML data source.

Another example is that a user gets real-estate data from several Internet sites that have slightly different schemas.

For example, the schemas shown in Figures 2 and 3 are both for house information [4].

```
<xsd:schema ...
  xmlns:house="http://www.greathomes.com">
  <xsd:element name="house"...>
    <xsd:element name="price"...>
    <xsd:element name="contact-phone"...>
        ......
```

Figure 2: XML Schema of greathomes.com

```
<xsd:schema ...
  xmlns:house="http://www.realestate.com">
  <xsd:element name="house"...>
    <xsd:element name="listed-price"...>
    <xsd:element name="phone"...>
        ......
```

Figure 3: XML Schema of realestate.com

When a user searches for houses, he/she first searches "greathomes.com" so he/she issues a query shown in Example 2 according to the schema in Figure 2.

Example 2. *Find the price and contact phone numbers of the houses for sale.*

```
FOR $g IN //house
RETURN
   <result>
      $g/price
      $g/contact-phone
   </result>
```

Later, the user wants to query data from "realestate.com" for the same information. Intuitively, he/she would hope that the query in Example 2 should work on data from "realestate.com" as well. However, in the data schema from "realestate.com" (Figure 3), "*price*" is called "*listed-price*" and "*contact-phone*" is called "*phone*", so the query in Example 2 will be rejected. On the other hand, the query will be executed if it is translated by the system to:

```
FOR $g IN //house
RETURN
   <result>
      $g/listed-price
      $g/phone
   </result>
```

From these two examples, we can see that a query evaluation and translation mechanism is needed when a user does not have enough knowledge about the data or is querying similar but yet slightly different data from multiple sources, for example, mining the information from the web.

When a query schema is not identical to the target data schema, the query should not be rejected up-front. There should be a mechanism that could test whether the query schema can be adjusted to the data schema, and if that is the case, rewrite the query according to the data schema.

The query and data schema mismatch problem has existed in traditional databases. Papakonstantinou et al. [13] addressed the issue of mismatch in the querying capability of different data sources, and they built a wrapper to reformulate the query thus resolved the mismatch. However, like the two E-Commerce examples shown above, when querying heterogeneous data sources on the Web, this mismatch problem occurs more often. Since XML is being widely used to represent data on the Web and as a common interoperable format in information integration systems, in this paper, we focus on solving the mismatch problem for XML query and data.

In the next section, we propose the *inclusion mapping* algorithm that matches a query schema Q to target data schema S. If Q and S mismatch, the algorithm will find if there exists a variation of Q, namely, Q', which is consistent in S.

3. THE INCLUSION MAPPING ALGORITHM

In this section we present an algorithm for matching schemas of queries against schemas of XML data. First, we formally introduce some preliminary notions. For simplicity, we refer to data schema as *schema* and query schema as *query*.

Let us denote by L the set of all legal XML tag names.

DEFINITION 3.1. *A query tree is a rooted, labeled tree* $Q = (V, E, \lambda_V, \lambda_E, r)$ *where:*

- V *is a set of nodes;*
- $E \subset V^2$ *is a set of edges;*
- $\lambda_V : V \rightarrow L \cup \{*\}$ *specifies a label for each node; the special label '*' means "unspecified";*
- $\lambda_E : E \rightarrow \{c, d\}$ *specifies a label for each edge: "c" means it is a child edge, and "d" means it is a descendant edge;*
- $r \in V$ *is the root of the tree.*

For example, the following XML query corresponds to the query tree in Figure 4.

EXAMPLE 3. *For articles ordered, find the recipients' name and city.*

```
FOR $p IN //purchaseOrder
   $a IN $p/articles/article
RETURN
   <result>
      $p/deliverTo/name
      $p/deliverTo/city
   </result>
```

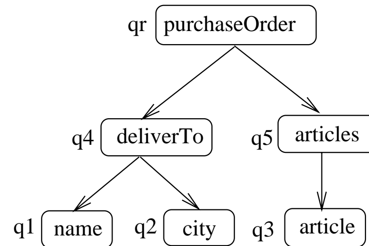

Figure 4: Query Tree

DEFINITION 3.2. *A* schema tree *is a rooted, labeled tree* $S = (V, E, \lambda_V)$ *where* V *is a set of vertices corresponding to elements or attributes,* E *is a set of child edges,* $\lambda_V : V \rightarrow L$ *maps each node to a tag name.*

Intuitively, a schema tree captures the set of all allowed paths from the root to a leaf of an XML document. For instance, an excerpt from the sample schema in W3C's XML Schema Part 0: Primer [10] can be represented in the schema tree shown in Figure 5.

```
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element name="purchaseOrder"
               type="PurchaseOrderType"/>

  <xsd:complexType name="PurchaseOrderType">
  <xsd:sequence>
    <xsd:element name="shipTo" type="USAddress"/>
    <xsd:element name="billTo" type="USAddress"/>
    <xsd:element name="items"  type="Items"/>
  </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="USAddress">
  <xsd:sequence>
  <xsd:element name="name"   type="xsd:string"/>
  <xsd:element name="city"   type="xsd:string"/>
  </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="Items">
  <xsd:sequence>
    <xsd:element name="item" minOccurs="0"
                 maxOccurs="unbounded"/>
    <xsd:element ref="comment" minOccurs="0"/>
  </xsd:sequence>
  </xsd:complexType>

</xsd:schema>
```

Each query tree and schema tree contains one or more *subtrees*.

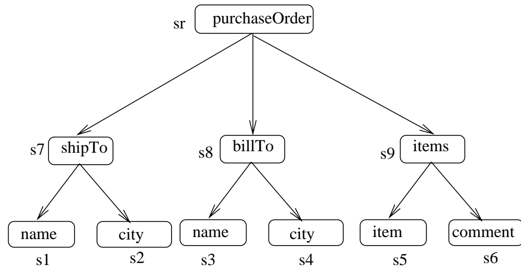

Figure 5: Schema Tree

Definition 3.3. *Given a query or a schema tree* $T = (V, E, ...)$, *we call* subtree *any subgraph* $T' = (V', E')$ *such that:*

- $V' = \{r', v_1, \ldots, v_k\}$ *where* r' *is a non-leaf node and* $v_1, \ldots, v_k$ *are all its children in* T*;*
- $v_1, \ldots, v_k$ *are all leaf nodes in* T*;*
- $E' \subseteq E$ *contains all the edges between the nodes in* V'*;*

The query tree in Figure 4 has two subtrees—one contains nodes q4, q1 and q2, the other contains nodes q5 and q3. And the schema tree in Figure 5 has three subtrees—one contains nodes s7, s1 and s2, one contains nodes s8, s3 and s4, and one contains nodes s9 s5 and s6. Indeed, the number of subtrees is the number of the distinct parents of the leaf nodes in a tree. The reason we introduce this notion is because that observation has been made [8] that most XML data schema trees have large numbers of nodes at the leaf level but not as many at other levels. Thus, the traditional use of the *fan out* factor of a tree is not desirable since that factor would usually be larger than the number of subtrees.

Given a query tree and a schema tree, we would like to find the "best match" between them. For example, if we examine the query tree in Figure 4 and the schema tree in Figure 5, intuitively we can guess that the best match would map `articles` to `items`, `deliverTo` to `shipTo` and so on. This intuition is based on the *semantic compatibility* between tag names and *structural compatibility* between the query tree and the schema tree. Let us make this notion more precise.

Definition 3.4. *Consider* L *the set of all legal XML tag names. A semantic compatibility measure is a function* $sim : L \cup \{*\} \times L \rightarrow [0, 1]$ *such that for all tag names* $w \in L$, $sim(w, w) = 1$ *and* $sim(*, w) = 1$*. The semantic compatibility factor between two tag names* w_1 *and* w_2 *is* $sim(w_1, w_2)$*.*

$$sim(w_1, w_2) = \frac{\Sigma_{i=1}^{n} 1 - \frac{\frac{l_{1i}+l_{2i}}{2}}{h}}{n} \quad (1)$$

where l_{1i} *and* l_{2i} *are the distance between the* $i-th$ *corresponding components of* w_1, w_2 *and their associated* synset *nodes in the WordNet [11] hierarchy, respectively;* h *is the number of hierarchical levels in WordNet;* n *is the number of components in the tag names.*

WordNet is a lexical database for English. It is organized by semantic relations. Synonymy is a lexical relation between word forms. For example, in WordNet, *boat* and $ferry$ are synonyms, their distance is 0. On the other hand, *boat* and *ship* both have *vessel* as direct hypernym, thus their distance is 1.

The semantic compatibility factor decreases when the distance increases. We set h to 10 since hierarchies in WordNet seldom go more than ten levels deep. When two words are synonyms in a WordNet hierarchy, i.e., the distance to their associated synset node $l_1 = 0$ and $l_2 = 0$, their semantic compatibility factor is 1; When both of the two words are 10 levels below their associated synset node, i.e., $l_1 = 10$ and $l_2 = 10$, their semantic compatibility factor is 0. If the tag names are multi-word terms, the average value of each component is used.

Definition 3.5. *Given a query tree* $Q = (V, E, \lambda_V, \lambda_E, r))$ *and a schema tree* $S = (W, F, \lambda_W)$*, an inclusion mapping is a function* $\mu : V \rightarrow W$ *such that:*

- *for every child query edge* (v, v'), $(\mu(v), \mu(v'))$ *is an edge in* S*;*
- *for every descendant query edge* (v, v') *there is a directed path* $(\mu(v) = w0, w1, \ldots, w_n = \mu(v'))$ *in* S*.*

An inclusion mapping induces a subgraph
$\mu(Q) = (\mu(V), \mu(F), \Lambda_W) \subseteq S$
containing all the images of the nodes in Q *together with all the nodes in the paths that are images of descendant query edges.*

In general, given a query tree and a schema tree, there may be many possible inclusion mappings. In order to distinguish among them, we define the *compatibility* of a mapping μ by taking into account both the semantic compatibility between the labels of the query tree and the labels of their corresponding images in the schema tree, and the structure of the trees.

Definition 3.6. *Given an inclusion mapping* μ *between a query tree* $Q = (V, E, \Lambda_V, \lambda_E, r))$ *and a schema tree* $S = (W, F, \Lambda_W)$*, we define the mapping compatibility* $co : V \rightarrow [0, 1]$ *recursively as follows:*

- *if* v *is a leaf node, then*

$$co(v) = sim(\lambda_V(v), \lambda_W(\mu(v))) \quad (2)$$

- *if* v *has children* $v_1, \ldots, v_k$*, then*

$$co(v) = \alpha \cdot \frac{\Sigma_{i=1}^{k} co(v_i)}{k} + (1-\alpha) \cdot sim(\lambda_V(v), \lambda_W(\mu(v))) \quad (3)$$

where $\alpha = \frac{k}{k+1}$ *is a fixed weight;*

The overall compatibility $co(\mu)$ *of a mapping* μ *is then defined as the compatibility of the root* $co(r)$*.*

The compatibility of a non-leaf node is the combination of its semantic compatibility factor and its children's compatibility factors. When a non-leaf node has only one child, the weight α is 0.5, i.e, both the child and the parent are considered equally. When the number of children increases, the role of the parent node becomes less and less important, i.e., $\lim_{k\rightarrow\infty} \alpha = 1$.

Finding the best inclusion mapping is a hard problem in general. One naive algorithm would be to enumerate all possible mappings and select the best one. This results in

a large number of mappings to be generated and compared. However, by observing real world XML data from different sources, we find that all sibling nodes are different from each other. We take advantage of this observation to limit the search space to a more manageable set while still guaranteeing the optimality of the solution. As a result, we can safely make the following assumption:

Assumption 3.1. **(Nonambiguity)**: *for any two sibling query nodes v and v', and every set of sibling schema nodes $W' = \{w_1, w_2, \ldots, w_k\}$, $\forall i, j$ $1 \leq i \leq k, 1 \leq j \leq k$, if $max(sim(v, w_i)) = max(sim(v', w_j))$ and $i = j$, then $v = v'$*

With these assumptions, we can process the query tree in any order and will always get the same result since among the leaf nodes of a subtree, which node we choose to start does not matter. Also comparing the compatibility factor of different matches just at the root level of the query tree is sufficient because the factor of the root node contains the contribution of all other nodes who are descendants of the root.

The pseudocode description of the algorithm is given in Algorithm 1.

We encode the nodes in both query and schema trees using the following *Dewey Decimal System*: the root is identified by Dewey ID 0, then its children from left to right by Dewey IDs $0.0, 0.1, \ldots, 0.k$. IDs for the children of any node are obtained by appending ".0", ".1", ..., ".m" to the Dewey ID of the current node. We chose this encoding for its good properties for tree navigation. For example the Dewey ID of the parent of any given node is the prefix to the last dot of the Dewey ID of the current node; this also means that one can easily check whether two nodes are siblings by comparing their prefixes. Finally, the level of a node is implicit in the Dewey ID: it is equal to the number of dots in the ID (the root has level 0).

The execution proceeds as follows: first, the nodes in the query and schema trees are sorted according to their Dewey IDs starting from the leaves, bottom-up and left to right (line 1). Then, the first query node (highest level, leftmost leaf) is matched against all schema nodes with the same level or higher and for each set of sibling schema nodes, only the best match is kept (lines 2–12). Subsequently, the remaining query nodes are matched for every one of the candidate matches for the previous query nodes. Every candidate combination is stored and its compatibility factor is maintained until the end (lines 13–37). When the root query tree is reached, the maximum compatibility combination is selected among all the candidate matches (lines 38–39).

The complexity of the algorithm is as follows. Suppose we have m subtrees in the query tree, and n subtrees in the schema tree, and assume all schema subtrees are possible matches to all query subtrees. For the first query subtree, there will be C_n^1 matches, for the second one C_{n-1}^1 matches, etc. The overall number of possible matches is

$$C_n^1 \times C_{n-1}^1 \cdots \times C_{n-i+1}^1 \times \cdots \times C_{n-m+1}^1$$
$$= n \times (n-1) \cdots \times (n-i+1) \times \cdots \times (n-m+1)$$
$$= O(n^m)$$

In summary, the number of possible mappings from the query tree to the schema tree is decided by the number of subtrees in the query tree and schema tree instead of the number of nodes in the trees. The execution time is poly-

Algorithm 1 Find Best Inclusion Mapping

```
1: sort the nodes in query tree
2: first query node q_1
3: for each schema subtree do
4:    for each sibling schema node s_k do
5:       if level of s_k ≥ level of q_1 then
6:          co_k = sim(tag(q_1), tag(s_k))
7:          find co_{q_1} = max_co among co_k and corresponding k
8:          store q_1, s_i, co_{q_1}, num_siblings = 1
9:       end if
10:   end for
11:   number of possible matches++
12: end for
13: for each query node q_i other than q_1 and the root do
14:   for each possible match do
15:      if q_i is parent of q_j where j < i then
16:         for each sibling schema node s_k do
17:            find the best match co_k
18:            calculate co_{q_i} using Equation 2
19:            store q_i, s_k, co_{q_i}, num_siblings = 1
20:         end for
21:      else if q_i is sibling of q_j where j < i then
22:         for each sibling schema node s_k do
23:            find the best match co_k
24:         end for
25:         co_{q_j} = co_{q_j} + co_k, num_siblings++
26:         store q_i, s_k
27:      else
28:         goto 31
29:      end if
30:   end for
31:   if processing a new query subtree then
32:      for each existing possible match do
33:         replicate the information for processed query
            subtrees, repeat the steps for the first query node
34:         number of possible matches = number of existing
            possible match × number of possible matches of
            q_i in schema tree
35:      end for
36:   end if
37: end for
38: handling root the same way as handling a parent node
39: find the mapping with the maximum compatibility factor
```

nomial to the number of subtrees in the schema and exponential to the number of subtrees in the query. In practice, most query trees are small in terms of the number of subtrees though the size of the schema tree could be very big. For example, almost all the queries in "XML Query Use Cases" [12] have only one subtree; very few (Q1, Q3 and Q5 in Section1.2 Use Case "TREE": Queries that preserve hierarchy) have two subtrees; none has more than two subtrees. So although the execution time grows with the number of query subtrees, since that number is only 1 or 2 in most cases, the running time is $O(n)$ or $O(n^2)$ where n is the number of subtrees in the data schema.

Next, we give a formal proof that the algorithm is correct in the sense that it will not leave out a possible mapping. We give the proof by Mathematical Induction Method.

Proof: Correctness of the Algorithm:

Suppose we have m nodes in the query tree Q, and n nodes in the schema tree S.

First, if $m = 1$, the mapping M_1 from Q to S is fixed.

Next, assume when $m = k$, the algorithm finds all mappings from Q to S.

Then, if $m = k+1$, the node q_{k+1} must be either a sibling of an existing node q_i in Q or the new root of Q.

(i) if node q_{k+1} is a sibling of node q_i in Q, then it belongs to the same subtree in Q as q_i. According to the Algorithm, the mappings are decided by the combination of subtrees in Q and S, so q_{k+1} does not introduce new mapping, though the compatibility factors need to be re-calculated for each mapping using Equation 3.

(ii) if node q_{k+1} is the new root of Q, it will not introduce new mapping because it will not create a new subtree. Again, Equation 3 will be used to re-calculate the new compatibility factors.

Similarly, we can apply the same induction method with m as a constant and n as a variant.

□

4. IMPLEMENTATION

4.1 The System Architecture

To provide a unified query schema over a collection of data sources, we built a mediator as shown in Figure 6. The mediator contains three parts: (1) the Query Matcher that finds the mapping between the query schema and the data schema using the *inclusion mapping* algorithm; (2) the Query Rewriter that translates the query to incorporate the data schema based on the mapping; (3) the Mapping Cache that stores previously generated mappings.

The Query Matcher works as follows: for each query to be performed against a set of data, it first fetches the schema information of the target data; Next, it checks the Mapping Cache, see if a mapping already exists. If not, the Query Matcher calculates the compatibility factor of the query schema vs. the data schema and record the mapping into the Mapping Cache.

As discussed in Section 3, we analyze two kinds of compatibility factors of a mapping: *semantic* and *structural*. The semantic compatibility factor of pairs of terms are precalculated using Equation 1 and checked by the Query Matcher at run time. For example, *send* and *ship* have a semantic compatibility factor of 1 since they are synonyms,

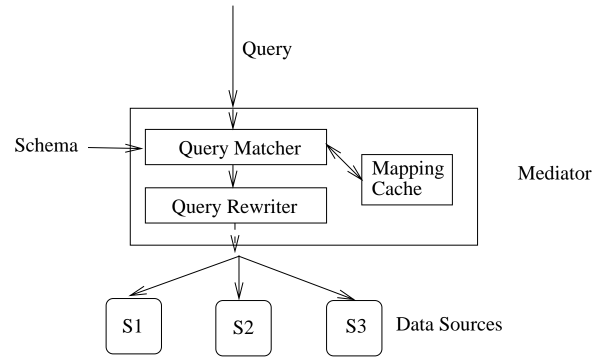

Figure 6: System Architecture

and *deliver* and *ship* have a factor of 0.85 because *deliver* is two levels below their synset node *transport*, and *ship* is one level below.

For multi-word terms, the semantic compatibility factor is the average of each component word's. For example, the terms, $firstName$ and $lastName$ have 0.5 since $sim(first, last) = 0$ and $sim(name, name) = 1$ but $name$ and $lastName$ have 1 since $sim(*, last) = 1$. Tentatively, we hard code in the information for abbreviated terms such as "DOB".

The structural compatibility of a mapping is calculated using Equation 3. For a leaf node, the structural compatibility factor is the same as its semantic compatibility factor. For a non-leaf node, the structural compatibility factor takes into consideration of both the structural information and the semantic similarity. Thus, when we say *compatibility factor of two nodes*, we are referring to their structural compatibility factor.

4.2 Empirical Results

In this section, we will present the empirical results for several queries.

The schema and data we perform the test upon are generated based on the purchase order schema "po.xsd" and data "po.xml" in [10].

The sample queries we used are as follows.

Query 1. *For each item ordered, find its name and price.*

```
FOR $i IN document("po.xml")
      /purchaseOrder/items/item
RETURN
   <result>
      $i/itemName
      $i/unitPrice
   </result>
```

Query 1 has only one subtree /purchaseOrder/items/item and two leaf elements itemName and unitPrice.

Query 2. *Find the items that were either ordered by somebody named "Smith" or shipped to somebody called that name.*

```
FOR $p IN document("po.xml")/purchaseOrder
   $s IN $p/*/name[contains(string(.), "Smith")]
RETURN
   <result>
      $p/items/item
   </result>
```

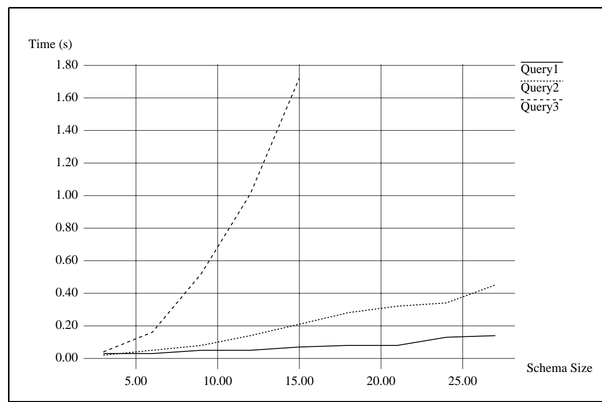

Figure 7: Matching time against number of schema subtrees

Query 2 has two subtrees. One is /purchaseOrder/items with a leaf element item; the other is /purchaseOrder/star with a leaf element name.

In this query, we use the "*star*" notation in the level below root to indicate that any element at that level might qualify the query.

Query 3. *For each item whose price is greater than $100, find the name of the person who ordered the item, to whom and to which city it is shipped.*

```
FOR $p IN document("po.xml")/purchaseOrder
   $i IN $p/items/item
   $b IN $p/billTo
   $s IN $p/shipTo
WHERE $i/unitPrice > 100
RETURN
   <result>
      $i/itemName
      $b/name
      $s/name
      $s/city
   </result>
```

Query 3 has three subtrees: /purchaseOrder/items/item, /purchaseOrder/billTo and /purchaseOrder/shipTo. All have two leaf elements.

We ran the tests on an IBM ThinkPad 600E with an Intel Pentium II 400MHz processor and 288MB of memory running Windows 2000. The implementation was done in Java.

We set the system default threshold at 0.9 and get the following results of applying the queries to the data.

Result for Query 1:

```
<result>
  <productName>Lawnmower</productName>
  <USPrice>148.95</USPrice>
  <productName>Baby Monitor</productName>
  <USPrice>39.98</USPrice>
</result>
```

Result for Query 2:

```
<result>
  <item partNum="872-AA">
    <productName>Lawnmower</productName>
    <quantity>1</quantity>
    <USPrice>148.95</USPrice>
    <comment>Confirm this is electric</comment>
  </item>
  <item partNum="926-AA">
    <productName>Baby Monitor</productName>
    <quantity>1</quantity>
    <USPrice>39.98</USPrice>
    <shipDate>1999-05-21</shipDate>
  </item>
</result>
```

Result for Query 3:

```
<result>
  <productName>Lawnmower</productName>
  <name>Robert Smith</name>
  <name>Alice Smith</name>
  <city>Mill Valley</city>
</result>
```

When we increase the threshold to 0.95, Query 1 is rejected, Query 2 and 3 are executed, and they return the same results as before. This is because the compatibility factor of the best mapping of Query 1 to the data schema is 0.917, less than 0.95.

Figure 7 shows the time spent to match the queries to thesize increasing data schema. The lines in the graph are polynomial when the size of the data schema increases, as our analysis of the algorithm in Section 3 predicted. For Query 1, the running time is linear since there is one subtree in the query; for Query 2, the running time is quadratic since it has two subtrees; and for Query 3, the running time is cubic since three subtrees are there. Moreover, the execution time increase sharply with the number of subtrees in the query schema but does not increase rapidly with the number of subtrees in the data schema for a given query.

The examples we have shown are queries without joins. When there is join in a query (from more than one XML document), different partial query trees will be generated for each source of data. Each partial query will be processed as a simple query without join. However, there may be cases that one partial query matches the target schema pretty well while another partial query matches badly. In this case, the decision of whether to continue the query rewriting process will be left to the user.

We have also run tests against DBLP data [3]. For example, the XPath query "/dblp/*/writer" returns all the <author> elements when the thresdhold is set at 0.9; and

it returns both <author> and <editor> elements when the thresdhold is set at 0.8. Indeed, the semantic compatibility factor between "writer" and "editor" is 0.8, "writer" and "author" is 1 since they are synonyms. However, there are also limitations. Another example is the XPath query "/dblp/inproceedings/title". It returns either <title> or <booktitle> because according to Definition 3.4, the semantic compatibility factor between "title" and "booktitle" is 1. We will need to improve the method to calculate the semantic compatibility factor in future work.

5. SUMMARY AND FUTURE WORK

We have presented an algorithm to match an XML query to the schema of target XML data. The algorithm identifies the part of the schema that has the maximum similarity with the query and computes an *inclusion mapping* from the query to the data schema. The similarity is determined by taking into consideration both the semantic similarity between element names and the structural compatibility between the query tree and schema tree.

The algorithm's running time is exponential in the number of subtrees of the query tree and polynomial in the number of subtrees of the schema tree. This makes the problem tractable in practice since the complexity of the queries is typically much smaller compared to the source schemas. For this reason, we believe our approach is a viable alternative to full-fledged schema mapping.

A limitation of our current algorithm is its inability to handle arbitrary descendant edges in the query tree (currently, it only handles descendant edges that emanate from the root). Descendant edges may increase the number of candidate matches to be examined. This becomes apparent even for single path queries that contain more than one descendant edges like, for example, `//a//b`. In this case, for every possible match s of the `b` node, the `a` node needs to be matched with all the ancestors of s in the schema tree.

As part of future work, we are planning to explore the possible reuse of mappings that were computed for a sequence of related queries; thus, the system could use the mappings stored in the Mapping Cache to infer a mapping for a new query.

Another direction for future work is to use a machine learning mechanism to decide semantic compatibility factor. The semantic compatibility factor between element names also depends on the contents of the text enclosed by a pair of tags. Sometimes, two pairs of tags may have different names but the contents enclosed by them belong to the same category. How to classify the content will be an interesting research problem.

6. REFERENCES

[1] S. Abiteboul, S. Cluet, T. Milo, P. Mogilevsky, J. Simon, and S. Zohar. Tools for data translation and integration. *IEEE Data Engineering Bulletin*, 22(1):3–8, 1999.

[2] C.-C. K. Chang and H. Garcia-Molina. Approximate query translation across heterogeneous information sources. In *Proceedings of the 26th International Conference on Very Large Data Bases*, pages 141–152, 2000.

[3] DBLP. *http://www.informatik.uni-trier.de/ ley/db/*.

[4] A. Doan, P. Domingos, and A. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 509 – 520, 2001.

[5] Y. Kanza and Y. Sagiv. Flexible queries over semistructured data. In *Proceedings of the 20th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 40–51, 2001.

[6] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic schema matching with cupid. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pages 49–58, 2001.

[7] R. J. Miller, L. M. Haas, and M. A. Hernndez. Schema mapping as query discovery. In *Proceedings of the 26th International Conference on Very Large Data Bases*, pages 77–88, 2000.

[8] A. Sahuguet. Everything you ever wanted to know about DTDs, but were afraid to ask. In *Proceedings of the 3rd International Workshop on the Web and Databases*, pages 171–183, 2000.

[9] H. Su, S. Padmanabhan, and M.-L. Lo. Identification of syntactically similar DTD elements for schema matching. In *Proceedings of the 2nd International Conference on Web-Age Information Management*, pages 145–159, 2001.

[10] W3C XML Schema Part 0: Primer. *http://www.w3.org/TR/xmlschema-0/*.

[11] WordNet - a Lexical Database for the English Language. *http://www.cogsci.princeton.edu/ wn/*.

[12] XML Query Use Cases. *http://www.w3.org/TR/xquery-use-cases/*.

[13] H. G.-M. Y. Papakonstantinou, A. Gupta and J. Ullman. A query translation schema for rapid implementation of wrappers. In *Proceedings of the 4th International Conference on Deductive and Object-Oriented Databases*, pages 99–113, 1995.