Detecting Structural Similarities between XML Documents

Sergio Flesca¹, Giuseppe Manco², Elio Masiari², Luigi Pontieri², and Andrea Pugliese¹,²

¹ DEIS, University of Calabria, Via Bucci 41c, 87036 Rende (CS) - Italy
email: {flesca,apugliese}@si.deis.unical.it
² ISI-CNR, Via Bucci 41c, 87036 Rende (CS) - Italy
email: {manco,masiari,pontieri}@isi.cs.cnr.it

Abstract. In this paper we propose a technique for detecting the similarity in the structure of XML documents. The technique is based on the idea of representing the structure of an XML document as a time series in which each occurrence of a tag corresponds to a given impulse. By analyzing the frequencies of the corresponding Fourier transform, we can hence state the degree of similarity between documents. The efficiency and effectiveness of this approach are compelling when compared with traditional ones.

1 Introduction

In this work we address the problem of identifying similarities between XML documents. In recent years, XML has gained increasing relevance as a means for exchanging information. As a matter of fact, most web applications deal with web data by translating them into XML format, and many commercial database systems (Oracle, IBM DB2) provide tools to deliver information in XML format and to store XML data. An interesting approach to efficiently store and retrieve XML documents is based on grouping together similar XML documents [5]. Algorithms for clustering documents according to their structural similarity could be effectively supported by our technique. Two relevant fields of application of our technique are the integration of semistructured data and the web site structural analysis. Indeed, grouping structurally similar documents can help in both recognizing sources providing the same kind of information and in presenting the information provided by a site. Several methods for detecting the similarity of XML documents [4, 3] have been recently proposed, that are based on the concept of edit distance [7] and use graph-matching algorithms to calculate a (minimum cost) edit script capable of transforming a document into another. Most of these techniques are computationally expensive, i.e. at least $O(N^3)$, where $N$ is the number of element of the two documents. Actually, the sub-optimal technique proposed in [4], works in quasi-linear time. However, all of them are concerned with the detection of changes occurring in XML documents rather than comparing them on the basis of their structural similarity. A different approach is adopted in [2], where a technique for measuring the similarity of a document versus a DTD is introduced. This technique exploits a graph-matching algorithm, which associates elements in the document with element definitions in the DTD. This approach does not seem to be directly applicable to cluster documents without any knowledge about their DTDs, and is not able to point out dissimilarities among documents referring to the same DTD.

Our aim and strategy are completely different. Indeed, we propose to represent the structure of an XML document as a time series, where each tag occurrence corresponds to an impulse. By analyzing the frequencies of the Fourier Transform of such series, we can state the degree of (structural) similarity between documents. As a matter of fact, the exploitation of the Fourier transform to check similarities among time series is not completely new (see, e.g., [1, 6]), and has been proven successful. The main contribution of our approach is the systematic development of an effective encoding scheme for XML documents, in a way that makes the use of the Fourier Transform extremely profitable. Efficiency and effectiveness of our approach are compelling when compared to the above mentioned ones, as we shall show in the rest of the paper.
2 Preliminaries

To our purposes an XML document can be considered as a tree of elements. A pair of tags delimits the area of the document that contains the information associated with an element. In turn, each element may contain further elements, as well as unstructured data (e.g. text). In our current approach, we deal with element attributes by regarding them as additional elements: an attribute element may contain further elements, as well as unstructured data (e.g. text). In our current approach, we deal with element attributes by regarding them as additional elements: an attribute element may contain further elements, as well as unstructured data (e.g. text).

Given an XML document \(d\) toy XML document \(X\) with name \(\tau\), we consider the toy XML document \(X\) with name \(\tau\). Moreover, \(\text{tnames}(d)\) denotes the set of all the distinct tag names appearing in \(d\). As an example, with regard to the document above, we have that \(\text{tnames}(d_0) = \{\text{xml}, \text{book}, \text{title}, \text{author}, \text{publisher}\}\) whereas \(\text{tags}(d_0) = \{(\tau_1, \text{<xml>}), (\tau_2, \text{<book>}), (\tau_3, \text{<ATTRIB@year>}), (\tau_4, \text{<ATTRIB@year>}), (\tau_5, \text{</title>}), (\tau_6, \text{</title>}), (\tau_7, \text{<author>}), (\tau_8, \text{</author>}), (\tau_9, \text{<author>}), (\tau_{10}, \text{</author>}), (\tau_{11}, \text{<publisher>}), (\tau_{12}, \text{/publisher>}), (\tau_{13}, \text{</book>}), (\tau_{14}, \text{</xml>})\}\), where each tag in \(\text{tags}(d_0)\) is denoted by a pair composed by its unique identifier and its textual representation. Furthermore, given a element \(el\) of an XML document \(d\), we denote by \(el_s\) the starting tag of \(el\) and with \(el_e\) the ending tag of \(el\). The definitions of the sets \(\text{tags}\) and \(\text{tnames}\) can be straightforwardly extended to deal with sets of XML documents.

Given an XML document \(d\), we define the skeleton of \(d\) as the sequence of tags appearing within \(d\), i.e. \(sk(d)\) is the sequence \([t_0, t_1, \cdots, t_n]\) such that \(t_i \in \text{tags}(d)\) \(\iff\) \(t_i \in sk(d)\) and \(t_i\) precedes \(t_j\) within \(d\) if and only if \(i < j\). In our approach, the skeleton of an XML document represents a description of the document structure. Moreover, it can be represented as an XML document with empty element content. For instance, the skeleton of the example document \(d_0\) can be represented as: \(<\text{xml}>\text{<book><ATTRIB@year>}/\text{ATTRIB@year}><\text{title}>\text{</title><author>}/\text{</author>}/\text{</publisher>}/</\text{xml}>\)\).

Finally, given a document \(d\) and a tag \(t\) in it, we define \(nest_d(t)\) as the set of the start tags \(el_s\) in \(d\) occurring before \(t\) and for which there is no end tag \(el_e\) matching \(el_s\) and appearing before \(t\). We also denote by \(l_t\) the nesting level of the tag \(t\), i.e. \(l_t = |nest_d(t)|\). For a given set \(D\) of documents, \(\text{maxdepth}(D)\) denotes the maximum nesting level of tags appearing in a document \(d\in D\).

3 Detecting Document Similarities

In this section we introduce our technique for detecting XML structural similarity. Intuitively, two documents are said to have a similar structure if they correspond in the names of the contained elements and in the way these elements are combined in the two documents.

The main idea of the approach is that of representing the skeleton of a document as a time series. More precisely, we can assume that we are visiting the tree-structure of an XML document in a depth-first, left-to-right way. As soon as we visit a node of the tree, we emit an impulse containing the information relevant to the tag. The resulting signal shall represent the original document as a time series. As a consequence, the comparison of two documents is done by looking at their corresponding signals. In the following, we first describe a technique for encoding the structure of a document into a time series, and next we define how to measure the similarity of such signals.

3.1 Document Structure Coding

A simple association of each tag name with a given number usually does not suffice to specify a suitable translation of a document. Indeed, the resulting time series is required to summarize the
main features of the document. Moreover, we need to represent these features in a suitable way, so that we can effectively distinguish two different documents by simply looking at their encodings. In this respect, we have considered several ways of encoding an XML document, obtained by specifying an encoding method for both the tags and the structure of the document. In a sense, a tag encoding corresponds to the analysis of the locality of a tag. On the other side, the nesting of different tags within the whole document provides an overall perspective: we look at the document as a globally uniform entity.

Tag Encoding. Given a set $D$ of XML documents, a function $\gamma$ from $\text{tags}(D)$ to $\mathbb{R}$ is a tag encoding function for $D$. We can assign a number $n$ to each tag in several ways: for instance, by generating it randomly, or using a hash function. However, a good tag encoding function should at least ensure injectivity, i.e., tags having different name are associated with two different numbers: for obvious reasons, collisions correspond to losing relevant information. A further desirable property is the capability to contextualize a given tag, i.e., to capture information about its neighbors.

We studied several tag encoding functions and in this work we will explain the one that guarantees a good compromise between efficiency and accuracy in the encoding. This tag encoding function, called direct tag encoding and denoted by $\gamma_d$, is built up as below specified.

Given a set $D$ of XML documents, we build a sequence of distinct tag names by considering a (randomly chosen) linear order on $\text{tnames}(D)$. Given an element $el$, the direct tag encoding simply associates each tag $el_s$ with the position of its name in the sequence. For the end tags, we consider two possible versions: either symmetric or null. A tag encoding function $\gamma$ is said to be symmetric iff for each document $d$ and for each element $el \in d$, $\gamma(el_s) = -\gamma(el_s)$; $\gamma$ is null if $\gamma(el_s) = 0$ (for each $d$ and $el$). For instance, the direct symmetric encoding of the example document $d_0$ is: $\gamma_d(\tau_1, \text{<xml>}) = 1$, $\gamma_d(\tau_2, \text{<book>}) = 2$, $\gamma_d(\tau_3, \text{<ATTRIB@year>}) = 3$, $\gamma_d(\tau_4, \text{</ATTRIB@year>}) = -3$, $\gamma_d(\tau_5, \text{<title>}) = 4$, $\gamma_d(\tau_6, \text{</title>}) = -4$, $\gamma_d(\tau_7, \text{<author>}) = \gamma_d(\tau_9, \text{<publisher>}) = 5$, $\gamma_d(\tau_8, \text{</author>}) = -5$, $\gamma_d(\tau_{11}, \text{<publisher>}) = 6$, $\gamma_d(\tau_{12}, \text{</publisher>}) = -6$, $\gamma_d(\tau_{13}, \text{</book>}) = -2$, $\gamma_d(\tau_{14}, \text{</xml>}) = -1$.

Notice that choosing a nondeterministic order on $\text{tnames}(D)$ excludes any possibility of exploiting context information. A possibility for improving the proposed scheme is that of imposing a significant order (e.g., by relating the position of each tag with its maximum nesting level within the documents). However, different approaches for representing tag context information may be defined, but we do not expose them here for the sake of conciseness.

Document Encoding. A document encoding is essentially a function that associates an XML document with a time series, representing the structure of the document. Let $D$ be the set of all the possible XML documents. A document encoding is a function that associates each $d \in D$ with a sequence of positive integers, i.e. $\text{enc}(d) = h_0, h_1, \ldots, h_n$.

A document encoding $\text{enc}$ is said to be without structural loss (WSL) iff for each pair of documents $d_1, d_2$, $\text{enc}(d_1) = \text{enc}(d_2)$ implies that $\text{sk}(d_1) = \text{sk}(d_2)$. Of course, the WSL property is desirable because it implies that we do not lose information about the document structure when encoding it, and we can reconstruct the structure of a document from its encoding. However, even if the WSL property holds, we are not guaranteed that we can effectively distinguish two documents by simply comparing their encodings. We have examined several document encoding functions to represent the document structure, that exploit a tag encoding function to define suitable sequences.

In this work we show only one of them, the multilevel encoding, which captures the structure of XML documents in a satisfactory manner. This encoding strategy weights each tag $t$ by considering its nesting level $l_t$ and using $\text{maxdepth}(D) - l_t$ as exponent of a fixed factor $B$, thus giving higher weights to elements appearing at higher levels of the document’s tree. In particular, consider a set $D$ of XML documents, a document $d \in D$ with $\text{sk}(d) = [t_0, \ldots, t_n]$ and a tag encoding function $\gamma$. Then, a multilevel encoding of $d$ is a sequence $[S_0, S_1, \ldots, S_n]$, where:

$$S_i = \gamma(t_i) \times B^{\text{maxdepth}(D) - l_i} + \sum_{t_j \in \text{nest}_i(t_i)} \gamma(t_j) \times B^{\text{maxdepth}(D) - l_j}$$
We usually set \( B = |\text{tnames}(D)| + 1 \) and choose the symmetric direct encoding as tag encoding function. In this way we avoid “mixing” together the contributions of different nesting levels and can reconstruct the path from the root to any tag by only considering the corresponding value in the encoded sequence. In fact, the summation in the right side of the formula above can be interpreted as the integer whose \((B+1)\)-base representation is the sequence of the tag codes in \( \{\gamma(t_j) \mid t_j \in \text{nest}_d(t_i)\} \), ordered by increasing nesting levels of the corresponding tags.

### 3.2 Similarity Measures

Faced with the above definitions, we can now detail the similarity measure for XML documents. As already mentioned, we suppose that we are visiting the tree-structure of an XML document \( d \) (in a depth-first, left-to-right way) starting from an initial time \( t_0 \). We also assume each node (tag) within the tree is found after a fixed time interval \( \Delta \). The total time spent to visit the document shall be \( N \Delta \) (where \( N \) is the cardinality of \( \text{tags}(d) \)). During the visit, as we find a start-tag we produce an impulse, that is assumed to stand until we reach the corresponding end-tag, and whose amplitude depends on the tag encoding and on the overall structure of the document (i.e., the document encoding \( \text{enc} \)).

As a result of the above physical simulation, the visit of the document produces a signal \( h_d(t) \), that usually changes its intensity, in the time interval \([t_0, t_0 + N \Delta]\). The intensity variations are directly related to the presence/absence of tags:

\[
h_d(t) = \begin{cases} \text{enc}(d)(k) & \text{if } t_0 + k \Delta \leq t < t_0 + (k + 1) \Delta \\ 0 & \text{if } t < t_0 \text{ or } t > t_0 + N \Delta \end{cases}
\]

Comparing two such signals, however, can be as difficult as comparing the original documents. First of all, comparing documents having different lengths requires to combine resizing and alignment, in a way that can be particularly hard to define. Stretching (or narrowing) signals is not a solution, since even with two signals having equal length the problem of defining the correct correspondences among the impulses can be extremely difficult. Finally, the intensity of a signal strongly depends on the encoding scheme adopted, that, in turn, may depend on the context (as in the case of the multilevel document encoding scheme). To this purpose it is particularly convenient to examine the Fourier transform of the signal, since it reveals much about the distribution and meaning of signal frequencies. Given a function \( h_d(t) \), we define a function \( \hat{h}_d(t) \) as the periodic extension of \( h_d(t) \). Hence, we are windowing the time series \( \hat{h}_d(t) \) of the document. In our particular case, we can compare the structure of two XML documents by exploiting the Fourier transforms. Given a document \( d \), we denote as \( \text{DFT}(\text{enc}(d)) \) the Discrete Fourier Transform of the time series resulting from the encoding. In particular, such a transform represents the frequencies ranging within the interval \([-0.5, 0.5]\) (obtained by setting \( \Delta = 1 \)).

In order to compare two documents \( d_i \) and \( d_j \), hence, we can work on their \( \text{DFT} \) transforms. In particular, a possibility \([1, 6]\) is to exploit the fact that, from Parseval’s theorem, energy is invariant in the transformation. However, a more effective discrimination can simply exploit the difference in the single frequencies: in a sense, we are interested (i) in abstracting from the length of the document, and (ii) in knowing whether a given subsequence (representing a subtree in the original XML document) exhibits a certain regularity, no matter where the subsequence is located within the signal. Following the above interpretation, we can measure the distance between two documents by computing the integral, over the given frequency range, of the magnitude difference of their transforms. In the discrete interpretation of the Fourier transformation, each point in the transform corresponds to a given frequency. Now, if \( d_i \) and \( d_j \) have different size, the corresponding (discrete) transform shows values corresponding to different frequencies. In order to approximate the above integral, we have to interpolate the corresponding values. In particular, we interpolate linearly the transform \( \text{DFT} \) and produce a new transform \( \tilde{\text{DFT}} \), having \( M = N_{d_i} + N_{d_j} - 1 \) points (where \( N_{d_i} = |\text{tags}(d_i)| \) and \( N_{d_j} = |\text{tags}(d_j)| \)).

More formally, the overall computation of the dissimilarity between documents can be done as follows. Let \( d_1 \) and \( d_2 \) be two XML documents, and \( \text{enc} \) be a document encoding, such that \( h_1 =
enc(d_1) and h_2 = enc(d_2). We define the Discrete Fourier Transform distance of the documents as the approximation of the difference of the magnitudes of the two signals:

$$
dist(d_1, d_2) = \left( \frac{1}{M/2} \sum_{k=1}^{M/2} \left( |DFT(h_1)(k)| - |DFT(h_2)(k)| \right)^2 \right)^{\frac{1}{2}}
$$

where $DFT$ is the interpolation of $DFT$ w.r.t. the frequencies appearing in both $h_1$ and $h_2$, and $M$ is the total number of points appearing in the interpolation. Alternative ways of comparing the documents can be defined. For example, we can choose to compare only a given number of values in the transforms, in the style of [1, 6].

Finally, a straightforward way to define a similarity measure is the following: given two documents $d_i$ and $d_j$, their similarity $s_{d_i,d_j}$ is computed as $1/(1 + dist(d_i, d_j))$.

It is worth noticing that the complexity of the computation of the above distance is mainly influenced by the computation of the $DFT$. It is easy to see that, when comparing two documents with length $N$, our method requires $O(N \log N)$, since computing their transforms is $O(N \log N)$.

## 4 Experimental Results

In this section we describe some experiments we performed to evaluate the effectiveness of the proposed method in measuring the structural similarity among XML documents. We assessed the validity of the proposed approach by comparing its results to some prior knowledge about the document similarities. Indeed, every dataset we considered consists of XML documents conforming to some previously chosen DTDs, so that it may be looked at as a set of structurally homogenous groups. For the sake of presentation we shall refer to any of such groups as document class or, more shortly, as class. We carried out several experiments on both real and synthesized datasets. In the following we describe some of the results on synthesized data, generated from the DTDs shown in fig. 1.

![Fig. 1. Example DTDs for Synthesized Data](image)

In order to build synthetic data sets, we implemented an XML document generator, which can produce a set of documents from a given DTD, according to various statistical models. Within each expression defining a DTD element, this system associates any occurrences of the operators $*$ and $+$ with a log-normal stochastic variable representing the length of the sequence that may be produced. Analogously, $|$ and $?$ operators are modelled by Bernoulli tests. The result of the experiments is a matrix representing the structural similarity degree for each pair of XML documents in the data set. In order to give an immediate and overall perception of the similarity relationships in the data set, we draw, in Figure 2, the similarity matrix as an image, where the grey level of each pixel is proportional to the value in the corresponding cell of the matrix.

Moreover, we introduce some summary measures to support simple quantitative analysis. Since we are interested in evaluating how much the similarity measure recognizes the a priori known class affinities, we compute all average intra-class similarities and all inter-class similarities. To this purpose, we report below a matrix $CS$, where the element $CS(i,j)$ contains the average of the similarity values corresponding to every pair of distinct documents such that the first belongs to the class $C_i$ and the second belongs to the class $C_j$. 
Fig. 2. Multilevel Encoding

<table>
<thead>
<tr>
<th></th>
<th>C₁</th>
<th>C₂</th>
<th>C₃</th>
<th>C₄</th>
<th>C₅</th>
</tr>
</thead>
<tbody>
<tr>
<td>C₁</td>
<td>0.9655</td>
<td>0.6418</td>
<td>0.8153</td>
<td>0.4822</td>
<td>0.3935</td>
</tr>
<tr>
<td>C₂</td>
<td>0.6418</td>
<td>0.9684</td>
<td>0.7485</td>
<td>0.4313</td>
<td>0.5037</td>
</tr>
<tr>
<td>C₃</td>
<td>0.8153</td>
<td>0.7485</td>
<td>0.9619</td>
<td>0.5402</td>
<td>0.4313</td>
</tr>
<tr>
<td>C₄</td>
<td>0.4822</td>
<td>0.4313</td>
<td>0.5402</td>
<td>0.9782</td>
<td>0.6817</td>
</tr>
<tr>
<td>C₅</td>
<td>0.3935</td>
<td>0.5037</td>
<td>0.4313</td>
<td>0.6817</td>
<td>0.9452</td>
</tr>
</tbody>
</table>

The results obtained by the encoding scheme here analyzed are very interesting. Indeed, they show that the method produces a neat distinction between elements belonging to a class and element outside that class. In addition, it is capable to capture structural affinities relating XML documents belonging to different classes. For instance, the results evidence a relatively high degree of similarity between the classes 1 and 3, whose DTDs exhibit quite similar structures.

5 Conclusions and Future Works

In this paper we showed an approach for measuring the structural similarity between XML documents. We proposed to represent the XML documents as time series and compute the structural similarity between two documents by exploiting the Discrete Fourier Transform of the corresponding signals. Experimental results showed the effectiveness of our approach, with particular reference to the proposed encoding schemes. Our technique could be refined by exploiting information retrieval techniques. In particular, the combination of the distance measure we propose with traditional techniques, such as Jaccard or Cosine similarity, can be extremely profitable. Furthermore an FFT-based distance measures different from the one introduced here could be used.

References

Evaluating Structural Similarity in XML Documents

Andrew Nierman and H. V. Jagadish
University of Michigan
{andrewdn, jag}@eecs.umich.edu

Abstract

XML documents on the web are often found without DTDs, particularly when these documents have been created from legacy HTML. Yet having knowledge of the DTD can be valuable in querying and manipulating such documents. Recent work (cf. [10]) has given us a means to (re-)construct a DTD to describe the structure common to a given set of document instances. However, given a collection of documents with unknown DTDs, it may not be appropriate to construct a single DTD to describe every document in the collection. Instead, we would wish to partition the collection into smaller sets of “similar” documents, and then induce a separate DTD for each such set. It is this partitioning problem that we address in this paper.

Given two XML documents, how can one measure structural (DTD) similarity between the two? We define a tree edit distance based measure suited to this task, taking into account XML issues such as optional and repeated sub-elements. We develop a dynamic programming algorithm to find this distance for any pair of documents. We validate our proposed distance measure experimentally. Given a collection of documents derived from multiple DTDs, we can compute pair-wise distances between documents in the collection, and then use these distances to cluster the documents. We find that the resulting clusters match the original DTDs almost perfectly, and demonstrate performance superior to alternatives based on previous proposals for measuring similarity of trees. The overall algorithm runs in time that is quadratic in document collection size, and quadratic in the combined size of the two documents involved in a given pair-wise distance calculation.

1 Introduction

The Extensible Mark-up Language (XML) is seeing increased use, and promises to fuel even more applications in the future. But many of these XML documents, especially those beginning to appear on the web, are without Document Type Descriptors (DTDs). In [10] the authors provide a method to automatically extract a DTD for a set of XML documents. They provide several benefits for the existence of DTDs. Given that more repositories of XML documents will exist in the future, methods will be needed to access these documents and perform queries over them, much as we do today with traditional database systems. Just as schemas are necessary in a DBMS for the provision of efficient storage mechanisms, as well as for the formulation and optimization of queries, the same is true for XML repositories and DTDs (which provide the schema). For instance, a DTD could allow a search to only access the relevant portions of the data, resulting in greater efficiency.

The algorithm in [10] is useful only when we apply it to a repository of XML documents where the repository is a homogeneous collection. If the collection includes structurally unrelated documents, then the DTD inferencing procedure will result in DTDs that are of necessity far too general and therefore not of much value. Ideally, the repository would be divided into groups of structurally similar documents first, and then the DTD inferencing mechanism could be applied individually to each of these groups.

In this paper we define a new method for computing the distance between any two XML documents in terms of their structure. The lower this distance, the more similar the two documents are in terms of structure, and the more likely they are to have been created from the same DTD. Crafting a good distance metric for this setting is somewhat difficult since two documents created from the same DTD can have radically different structures (due to repeating and optional elements), but we would still want to compute a small distance between these documents. We account for this by introducing edit operations that allow for the cutting and pasting of whole sections of a document. Using our resulting pair-wise distance measure, we show that standard clustering algorithms do very well at pulling together documents derived from the same DTD.

2 Background

2.1 XML Data Model

An XML document can be modeled as an ordered labeled tree [9]. Each node in this tree corresponds to an element in the document and is labeled with the element tag name. Each edge in this tree represents inclusion of the element corresponding to the child node under the element corresponding to the parent node in the XML file.

XML documents may also have hyper-links to other documents. Including such links in the model gives rise to a graph rather than a tree. Such links can be important in actual use of the XML data. However, they are not important as far as the structure of the document at hand, and hence we will not consider them further in this paper.

A DTD provides rules that define the elements, attributes associated with elements, and relationships among
elements, that may occur in an XML document. DTDs have
the expressive power of regular languages: elements may
be required, optional, or may be repeated an arbitrary num-
ber of times. Attributes may also be required or optional.

2.2 Attributes in the Data Model

Elements in XML can have attributes, and these attributes
can play an important role in the DTD determination prob-
lem we are attempting to tackle. The traditional DOM la-
belled ordered tree has one node for every element in the
document: attributes adorn the node corresponding to the
element of which they are attributes. To incorporate at-
tributes into our distance calculation, we create an addi-
tional node in the tree for each attribute, and label it with
the name of the attribute. These attribute nodes appear as
“children” of the node that they adorned in the DOM rep-
resentation, sorted by attribute name, and appearing before
all sub-element “siblings”.

In short, we represent each XML document as a labeled
ordered tree with a node corresponding to each element and
to each attribute. We do not represent the actual values of
the elements or attributes in the tree – we are only interested
in the structural properties of the XML file.

2.3 Related Work

There is considerable previous work on finding edit dis-
tances between trees [5–8, 13–17]. Most algorithms in
this category are direct descendants of the dynamic pro-
gramming techniques for finding the edit distance between
strings [12]. The basic idea in all of these tree edit distance
algorithms is to find the cheapest sequence of edit opera-
tions that can transform one tree into another.

A key differentiator between the various tree-distance
algorithms is the set of edit operations allowed. An early
work in this area is by Selkow [13], and allows inserting
and deleting of single nodes at the leaves, and relabeling of
nodes anywhere in the tree. The work by Chawathe in [5]
utilizes these same edit operations and restrictions, but is
targeted for situations when external memory is needed
to calculate the edit distance. There are several other ap-
proaches that allow insertion and deletion of single nodes
anywhere within a tree [14–17].

Expanding upon these more basic operators, Chawathe,
et. al. [7] define a move operator that can move a subtree as
a single edit operation, and in subsequent work [6] copying
(and its inverse, gluing) of subtrees is allowed. These two
operations bear some resemblance to the insert subtree and
delete subtree operations that are used in this paper, but the
approaches in [6, 7] are heuristic approaches and the algo-
rithm in [6] operates on unordered trees, making it unsuit-
able for computing distances between XML documents.

3 Tree Edit Distance

Two XML documents produced from the same DTD can
have very different sizes on account of optional and repeat-
ing elements. Any edit distance metric that permits change
to only one node at a time will necessarily find a large dis-
tance between such a pair of documents, and consequently
will not recognize that these documents should be clustered
together as being derived from the same DTD. In this sec-
tion, we develop an edit distance metric that is more indica-
tive of this notion of structural similarity. First we present
a few supporting definitions.

3.1 Basic Definitions

Definition 3.1 [Ordered Tree] An ordered tree is a
rooted tree in which the children of each node are ordered.
If a node x has k children then these children are uniquely
identified, left to right, as x1, x2, …, xk.

Definition 3.2 [First-Level Subtree] Given an ordered
tree T with a root r of degree k, the first-level sub-
trees, T1, T2, …, Tk of T are the subtrees rooted at
r1, r2, …, rk.

Definition 3.3 [Labeled Tree] A labeled tree T is a tree
that associates a label, λ(x), with each node x ∈ T. We let
λ(T) denote the label of the root of T.

Definition 3.4 [Tree Equality] Given two ordered
labeled trees A and B, and their first-level subtrees
A1, A2, …, An and B1, B2, …, Bn, A = B if: λ(A) = λ(B),
m = n, and i = j ⇒ Ai = Bj, for 0 ≤ i ≤ m, 0 ≤ j ≤ n.

3.2 Tree Transformation Operations

We utilize five different edit operations in the construc-
tion of our algorithm. Given a tree T with λ(T) = l and first-
level subtrees T1, …, Tm, the tree transformation operations
are defined as follows:

Definition 3.5 [Relabel] RelabelT(lnew) is a relabel
operation applied to the root of T that yields the tree T′ with
λ(T′) = lnew and first-level subtrees T1, …, Tm.

Definition 3.6 [Insert] Given a node x with degree 0,
InsertT(x, i) is a node insertion operation applied to T
at i that yields the tree T′ with λ(T′) = l and first-level
subtrees T1, …, Ti, x, Ti+1, …, Tm.

Definition 3.7 [Delete] If the first-level subtree Ti is a
leaf node, DeleteT(Ti) is a delete node operation applied
to T at i that yields the tree T′ with λ(T′) = l and first-
level subtrees T1, …, Ti−1, Ti+1, …, Tm.

Definition 3.8 [Insert Tree] Given a tree A,
InsertTreeT(A, i) is an insert tree operation applied
to T at i that yields the tree T′ with λ(T′) = l and first-
level subtrees T1, …, Ti−1, Ti, Ti+1, …, Tm.

Definition 3.9 [Delete Tree] DeleteTreeT(Ti) is a
delete tree operation applied to T at i that yields
the tree T′ with λ(T′) = l and first-level subtrees
T1, …, Ti−1, Ti+1, …, Tm.
Associated with each of these edit operations is a non-negative cost. Our algorithms work with general costs, but in this paper we restrict our presentation and experimentation to constant (unit) costs.

### 3.3 Allowable Edit Sequences

The usual way in which the edit distance is found between two objects is to consider alternative sequences of edit operations that can transform one object into the other. The cost of the operations in each sequence is considered, and the lowest cost sequence among these defines the edit distance between the two objects. In our case, rather than considering all possible sequences of edit operations, we restrict ourselves to all “allowable sequences” of edit operations. We do this both for computational reasons, as well as to improve our results in the XML domain.

**Definition 3.10** [Minimum Edit Distance Cost ($\delta$)]

Given any trees $A$ and $B$ and the set $\Xi$ of all allowable sequences of edit operations that when applied to $A$ will yield a tree equal to $B$, we let $\delta(A, B)$ denote the minimum of the sums of the costs of each sequence in $\Xi$.

**Definition 3.11** [Allowable]

A sequence of edit operations is **allowable** if it satisfies the following two conditions:

1. A tree $P$ may be inserted only if $P$ already occurs in the source tree $A$. A tree $P$ may be deleted only if $P$ occurs in the destination tree $B$.
2. A tree that has been inserted via the $\text{InsertTree}$ operation may not subsequently have additional nodes inserted. A tree that has been deleted via the $\text{DeleteTree}$ operation may not previously have had (children) nodes deleted.

The first restriction limits the use of the insert tree and delete tree operations to when the subtree that is being inserted (or deleted) is shared between the source and destination tree. We can only insert (delete) subtrees that are already “contained in” the source (destination) tree. A pattern tree $P$ is said to be $\text{containedIn}$ tree $T$, if all nodes of $P$ occur in $T$, with the same parent/child edge relationships and same sibling order; additional siblings may occur in $T$, even between sibling nodes in the embedding of the pattern tree. This allows for matching of trees when optional elements are used in DTDs. See Figure 1 for some examples of the $\text{containedIn}$ relation, where a pattern tree $P$ is potentially $\text{containedIn}$ various other trees. Without this first restriction on allowable sequences of edit operations, one could delete the entire source tree in one step and insert the entire destination tree in a second step – totally defeating the purpose of the insert tree and delete tree operations.

The second restriction provides us with an efficient means for computing the costs of inserting and deleting the subtrees found in the destination and source trees, respectively. This procedure is outlined in the next section.

![Figure 1: Examples of the containedIn Procedure](image)

### 4 Algorithm

#### 4.1 Dynamic Programming Formulation

Dynamic programming is frequently used to solve minimum edit distance problems. In determining the distance between a source tree, $A$, and a destination tree, $B$, the key to formulating the problem using dynamic programming is to first determine the cost of inserting every subtree of $B$, and the cost of deleting every subtree of $A$.

When determining the cost of inserting a subtree $T_i$, this could possibly be done with a single $\text{InsertTree}$ operation (if it is allowable), or with some combination of $\text{InsertTree}$ and $\text{Insert}$ operations. There is a cost associated with each possible sequence of $\text{InsertTree}$ and $\text{Insert}$ operations that result in the construction of the subtree $T_i$. The minimum of these costs is denoted as the $\text{graft}$ cost of $T_i$, or $\text{Cost}_{\text{Graft}}(T_i)$. A $\text{prune}$ cost is defined similarly for the minimum cost sequence of $\text{Delete}$ and $\text{DeleteTree}$ operations needed to remove a subtree.

Due to the constraints specified in definition 3.11 for an allowable sequence, we have a simple and efficient bottom-up procedure for computing the graft cost, $\text{Cost}_{\text{Graft}}$. At each node $v \in B$ we calculate the cost of inserting the single node $v$ and add the graft cost of each child of $v$, we call this sum $d_0$. We also check whether the pattern tree $P$, which is the subtree rooted at $v$, is $\text{containedIn}$ the source tree $A$. If $\text{containedIn}(P, A)$ is true, we compute the insert tree cost for $P$, we call this sum $d_1$. The graft cost for the subtree rooted at $v$ is the minimum of $d_0$ and $d_1$. Prune costs are computed similarly for each node in $A$.

Given a source tree $A$ and a destination tree $B$, we can determine the minimum cost of transforming $A$ into $B$ using the operators defined in section 3.2, and the notion of allowable sequences in section 3.3. This dynamic programming algorithm is shown in figure 2. Pre-computed costs for the graft and prune costs are used in lines 8 and 10, and in the nested loops at lines 16 and 17 of the algorithm.

#### 4.2 Complexity

In this section we analyze the complexity of computing our edit distance measure between a source tree $A$ and a destination tree $B$. There are two stages to the algorithm. In the first stage, all the graft and prune costs are pre-computed. In the second stage, we use these pre-computed values to compute the actual edit distance, as given in figure 2.
Given that the maximum degree of any node in $B$ is assumed constant, independent of the sizes of the trees, the complexity for computing graft costs is $O(|B|)$ and the overall complexity of this stage is $O(|A||B|)$. Having computed these containedIn relations, the graft and prune costs can be calculated, as in section 4.1, by simply performing post-order traversals of $B$ and $A$, respectively, so the complexity of these operations is simply $O(|B|)$ and $O(|A|)$, and the overall complexity of this stage is $O(|A||B|)$.

4.2.2 Stage Two – Dynamic Programming ($editDistance$)

The $editDistance$ procedure in figure 2 is called once for each pair of vertices at the same depth in the input trees $A$ and $B$. This results in a complexity of $O(|A||B|)$ [13].

4.2.3 Overall Complexity

$O(|A||B|)$ is the time complexity for both the pre-computation phase, and the dynamic programming phase, so $O(|A||B|)$ is the overall complexity of our algorithm to compute structural edit distance between two trees $A$ and $B$. This linear dependence on the size of each tree (and quadratic dependence on the combined size of the two trees) is borne out in the experimental results shown in Section 5.3.

5 Experimental Evaluation

The goal of our work is to find documents with structural similarity, that is, documents generated from a common DTD. We apply a standard clustering algorithm based on the distance measures computed for a given collection of documents with known DTDs. For any choice of distance metric, we can evaluate how closely the reported clusters correspond to the actual DTDs.

5.1 Setup

5.1.1 Algorithms Used

In addition to our edit distance measure, we evaluate two measures proposed previously in the literature for tree edit distance – which we refer to as Chawathe [5] and Shasha [14] respectively, and a third non-structural baseline metric. We report results in this section for these three measures in addition to our own.

The Chawathe measure – In [5] an algorithm is presented for computing differences between hierarchically structured data such as XML. Disregarding the work’s contribution towards efficient use of secondary storage, our algorithm can be seen as a strict generalization of this approach. Specifically, if we disallow tree insertions and deletions in our measure, we would obtain exactly the Chawathe measure. The complexity of this approach is $O(|A||B|)$, when finding the minimum edit distance between the trees $A$ and $B$.

The Shasha Measure – Dennis Shasha and Jason Wang propose a tree edit distance metric in [14] that permits the addition and deletion of single nodes anywhere in the tree, not just at the leaves. However, entire subtrees cannot be
inserted or deleted in one step. The complexity of this approach is \( O(|A| |B| \text{depth}(A) \text{depth}(B)) \).

**Tag Frequency** \( (\delta_{\text{freq}}) \) – A good question to ask is whether all this complex tree structure based difference is a good thing to do in the first place. How about a simple measure that looks at the count of each type of label in the two documents, and adds up the absolute values of the differences? By utilizing a simple hash data structure for the element names and the frequencies, we can compute the tag frequency distance, \( \delta_{\text{freq}} \), between two trees \( A \) and \( B \) in \( O(|A| + |B|) \).

### 5.1.2 Data Sets Used

We performed experiments on both real and synthetic data sets. For a real data set, we used XML data obtained from the online XML version of the ACM SIGMOD Record [1]. Specifically, we sampled documents from each of the following DTDs: ProceedingsPage.dtd, Index-TermsPage.dtd, and OrdinaryIssuePage.dtd.

We also utilized synthetic data generated in an automated fashion from real DTDs. Real-world DTDs were obtained online from [2, 3] and an XML document generator [4] that accepts the DTDs as input was used to generate the XML documents. We varied the following two key parameters to generate repositories:

**MaxRepeats** The maximum number of times a child element node will appear as a child of its parent node (when the * or + option is used in the DTD). A value between 0 and MaxRepeats is chosen randomly for each repeating node (rather than once for the entire document). The greater this number, the greater the fanout and also the greater the variability in fanout.

**Attribute Occurrence Probabilities** There are both required and optional attributes specified in a DTD. We let \( \text{Prob}_{\text{Attribute}} \) equal the probability that an optional attribute will occur.

We experimented with values for **MaxRepeats** in the range [2,12]. Also, we tested the following values for the attribute occurrence probabilities: \( \text{Prob}_{\text{Attribute}} \in \{.1,.25,.5,.75,.9,1\} \). In this paper, we present a representative sampling of these tests with the following synthetic data sets:

- **Data Set 1**: \( \text{MaxRepeats} = 4 \), \( \text{Prob}_{\text{Attribute}} = .75 \);
- **Data Set 2**: \( \text{MaxRepeats} = 4 \), \( \text{Prob}_{\text{Attribute}} = 1 \) (attributes always appear);
- **Data Set 3**: \( \text{MaxRepeats} = 8 \), \( \text{Prob}_{\text{Attribute}} = .75 \);
- **Data Set 4**: \( \text{MaxRepeats} = 8 \), \( \text{Prob}_{\text{Attribute}} = 1 \).

All operator costs were set equal to 1 for all of the experiments that we present in this paper.

1. The DTDs that were used were: HealthProduct.dtd, blastxml.dtd, dri.dtd, flights.dtd, fixxml.dtd, roamops-phonebook.dtd, vcard.dtd, and dsml.dtd.
2. For all values of \( \text{Prob}_{\text{Attribute}} \), not equal to 1, there were no appreciable differences in the results, and we simply show the results for \( \text{Prob}_{\text{Attribute}} = .75 \).
3. Small changes in operator costs did little to affect overall clustering accuracy. These results are not presented due to lack of space.

### 5.1.3 Computing Environment

These tests were done on an IBM RS6000 with dual processor 604e PowerPC Processors, running at 332 MHz. All approaches, except the Shasha measure, were implemented by us in Java. The Shasha measure is implemented in C and thus the timing results cannot be directly compared with the other methods.

### 5.2 Clustering

Due to lack of space we do not present the distances obtained from comparing all pairs of documents to each other, rather we simply present the clustering results that were obtained using these distances. We utilize well-known techniques in hierarchical agglomerative clustering [11] (although any form of clustering could be used).

The end result can be represented visually as a tree of clusters called a **dendrogram**. The dendrogram shows the clusters that were merged together, and the distance between these merged clusters (the horizontal length of the branches is proportional to the distance between the merged clusters). Two example dendrograms can be seen in figure 3.

Clustering algorithms require knowledge of the distance between any pair of clusters, including single document “clusters”. For this purpose, we use the Unweighted Pair-Group Averaging Method (or UPGMA). The distance between clusters \( C_i \) and \( C_j \) is computed as follows:

\[
\text{Distance}(C_i, C_j) = \frac{\sum_{k=1}^{C_i} \sum_{l=1}^{C_j} \delta(doc_k^{C_i}, doc_l^{C_j})}{|C_i||C_j|}
\]

Where \( |C_i| \) is the number of XML documents contained in cluster \( C_i \) and \( doc_k^{C_i} \) is the \( k \)th XML document in the cluster \( C_i \).

In order to compare the hierarchical clustering results, we introduce our notion of a “mis-clustering”. Given a dendrogram, the number of mis-clusterings is equal to the minimum number of documents in the dendrogram that would have to be moved, so that all documents from the same DTD are grouped together. A small sample clustering is shown in figure 3: in this example our approach has no mis-clusterings, while the Chawathe approach has three mis-clusterings.

A summary of the number of mis-clusterings, for each of the data sets, is found in table 1. Our approach performs better than the competing approaches for each of the data sets (and in fact, in the underlying data, there is no average intra-DTD distance that is lower than our approach, and no average inter-DTD distance that is higher). Our approach does better when all attributes are forced to appear, since there would be more subtrees that “look” the same in documents generated from the same DTD, and the **containedIn** procedure would return true more often.
b) Our Approach

Figure 3: Sample Clustering Results for SIGMOD Record.

<table>
<thead>
<tr>
<th></th>
<th>Data Set 1</th>
<th>Data Set 2</th>
<th>Data Set 3</th>
<th>Data Set 4</th>
<th>SIGMOD Record</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our Approach</td>
<td>10</td>
<td>2</td>
<td>11</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>Chawathe</td>
<td>16</td>
<td>8</td>
<td>30</td>
<td>25</td>
<td>3</td>
</tr>
<tr>
<td>Shasha</td>
<td>16</td>
<td>9</td>
<td>32</td>
<td>39</td>
<td>3</td>
</tr>
<tr>
<td>Tag Frequency</td>
<td>22</td>
<td>21</td>
<td>35</td>
<td>40</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 1: Number of Mis-Clusterings for Each Approach

5.3 Timing Analysis

Our algorithm appears more complex conceptually, however, its asymptotic time complexity ($O(|A||B|)$) is the same as the Chawathe algorithm, and slightly better than Shasha’s algorithm ($O(|A||B|\text{depth}(A) \text{depth}(B))$). We are asymptotically worse than the δfree approach, which is $O(|A| + |B|)$, but this approach performs poorly in terms of clustering the documents. The formulae are verified experimentally, and we show the timing results for our approach in figure 4. The time to find the edit distance between pairs of trees of various sizes grows in an almost perfect linear fashion with tree size (of each tree). The corresponding times for the Chawathe technique were smaller by (only) a factor of 1.6 on average.

Figure 4: Our Approach - Timing Results (to compute pairwise distance) for Various Tree Sizes

6 Conclusion

XML is becoming all-pervasive, and effective management of XML data is a high priority. The applicability of many database techniques to XML data depends on the existence of DTDs (or schema) for this data. In the laissez-faire world of the Internet, though, we frequently have to deal with XML documents for which we do not know the schema. While there has been previous work on deducing the DTD for a collection of XML documents, such algorithms depend critically on being given a relatively homogeneous collection of documents in order to determine a meaningful DTD.

In this paper we have developed a structural similarity metric for XML documents based on an “XML aware” edit distance between ordered labeled trees. Using this metric, we have demonstrated the ability to accurately cluster documents by DTD. In contrast, we have shown that several other measures of similarity do not perform as well, while requiring approximately the same amount of computation.

References

Obtaining More Answers From Information Integration Systems

Gösta Grahne and Victoria Kiricenko
Department of Computer Science, Concordia University
Montreal, Quebec, Canada H3G 1M8
{grahne,kiricen}@cs.concordia.ca

Abstract
The current generation of rewriting-algorithms in source-centric (local-as-view) information integration systems all produce a reformulated query that retrieves what has been thought of as “the best obtainable” answer, given the circumstances that the source-centric approach introduces incomplete information into the virtual global relations. This “best obtainable” answer does not however allow partial information. We define the semantics of partial facts, and provide two methods for computing partial answers. The first method is tableau-based and is a generalization of the “inverse-rules” approach. The second method is a generalization of the rewriting approach, and is based on partial containment mappings introduced in the paper.

1 Introduction
Information Integration systems aim to provide a uniform query interface to multiple heterogeneous sources. One of the ways to view these systems is to postulate a global schema (called a world view) that provides a unifying data model for all the information sources (see e.g. [Len02]). A query processor is in charge of reformulating queries written in terms of this global schema to queries on the appropriate sources and assembling the answers into a global answer. Each source is modeled as a materialized view defined in terms of the global relations, which are virtual.

To illustrate the problem with current information integration methodologies let us consider a simple example. Suppose the user issues the query

\[ Q(Pname, Email, Office, Area) \leftarrow Prof(Pname, Email, Office, Area), Dept(Pname, Dname) \]

That is, the user is interested in obtaining all information available about professors in the compsci department. Since the information integration system does not have a way to get tuples for the subgoal \( Prof \) it would produce an empty rewriting and, thus, an empty answer for the user.
From the point of view of the user, it would be much more useful to get at least partial information about professors in the compsci department. This is feasible, the query could be rewritten as union of the following unsafe conjunctive queries.

\[ Q_1(\text{Pname}, \text{Email}, X, Y) \leftarrow S_1(\text{Pname}, \text{Email}), S_4(\text{Pname}, \text{compsci}) \]

\[ Q_2(\text{Pname}, X, \text{Office}, Y) \leftarrow S_2(\text{Pname}, \text{Office}), S_4(\text{Pname}, \text{compsci}) \]

\[ Q_3(\text{Pname}, X, Y, \text{Area}) \leftarrow S_3(\text{Pname}, \text{Area}), S_4(\text{Pname}, \text{compsci}) \]

The unrestricted variables \( X \) and \( Y \) represent unknown values. The answer can then be presented to the user as a table with some values missing, for example as the table below. (The contents of the table will, obviously, depend on the data provided by the sources.)

<table>
<thead>
<tr>
<th>Pname</th>
<th>Email</th>
<th>Office</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>Murphy</td>
<td><a href="mailto:murphy@cs.toronto.edu">murphy@cs.toronto.edu</a></td>
<td>⊥</td>
<td>⊥</td>
</tr>
<tr>
<td>Murphy</td>
<td>⊥</td>
<td>SF322</td>
<td>⊥</td>
</tr>
<tr>
<td>Smith</td>
<td>⊥</td>
<td></td>
<td>DB</td>
</tr>
<tr>
<td>Jones</td>
<td><a href="mailto:jones@cs.concordia.ca">jones@cs.concordia.ca</a></td>
<td>⊥</td>
<td>⊥</td>
</tr>
<tr>
<td>Brown</td>
<td>⊥</td>
<td></td>
<td>AI</td>
</tr>
</tbody>
</table>

This aspect of information integration has not been studied in the current literature despite its importance especially in the case of the world wide web, where we can expect lots of partially overlapping sources. In this paper we define answers containing null values, and we give two algorithms for computing them. One algorithm is a generalization of the “inverse-rules” approach [DG97], and involves explicitly computing a syntactic representation of the set of global databases implicitly defined by the sources. The other algorithm is a generalization of the rewriting technique (see e.g. [LMSS95], [Ull97]). This algorithm reformulates a query in terms of the source relations, and, thus, avoids inverting the entire source collection.

2 Conjunctive Queries and Projection-Containment

Let \( \text{rel} \) be a countably infinite set \( \{R, S, \ldots, R_1, S_1, R_2, S_2, \ldots\} \) of relation names, let \( \text{dom} \) be a countably infinite set of constants, and let \( \text{var} \) be a countably infinite set of variables. Constants will be denoted by lower case letters and variables by upper case letters.

Associated with each relation name \( R \) is a positive integer \( \text{arity}(R) \), which is the arity of \( R \). A fact over \( R \) is an expression of the form \( R(a_1, \ldots, a_k) \), where \( k = \text{arity}(R) \), and each \( a_i \) is in \( \text{dom} \).

Let \( \mathbf{R} = \{R_1, R_2, \ldots, R_n\} \) be a set of relation names. A finite set of relation names will sometimes also be called a schema. A database \( d \) over \( \mathbf{R} \) is a finite set of facts, each fact being over some \( R_i \in \mathbf{R} \).

An atom over a relation name \( R \) is an expression of the form \( R(e_1, \ldots, e_k) \), where \( k = \text{arity}(R) \), and each \( e_i \) is either a constant in \( \text{dom} \) or a variable in \( \text{var} \).

A conjunctive query \( \varphi \) (over \( \mathbf{R} \)) has the form

\[ \text{head}(\varphi) \leftarrow \text{body}(\varphi), \]

where \( \text{body}(\varphi) \) is a set of atoms \( b_1, b_2, \ldots, b_n \), each over a relation name in \( \mathbf{R} \), and \( \text{head}(\varphi) \) is an atom over an answer relation name not in \( \mathbf{R} \). We assume that all variables occurring in \( \text{head}(\varphi) \) also occur in \( \text{body}(\varphi) \), i.e., that the query \( \varphi \) is safe. The variables occurring in \( \text{head}(\varphi) \) are the distinguished variables of the query, and all the others are existential variables.

A conjunctive query \( \varphi \) can be applied to a database \( d \) over \( \mathbf{R} \), resulting in a set of facts

\[ \varphi(d) = \{ \sigma(\text{head}(\varphi)) : \sigma(\text{body}(\varphi)) \subseteq d \text{ for some valuation } \sigma \}. \]
A valuation $\sigma$, is formally a finite partial mapping from $\var{\text{var}} \cup \text{dom}$ to $\text{dom}$ that is the identity on $\text{dom}$. Valuations, like $X_i \mapsto a_i$, for $i \in [1,p]$, will usually be given in the form $\{X_1/a_1, \ldots, X_p/a_p\}$. The identity on constants is omitted in this notation.

The notion of query containment enables comparisons between different reformulations of queries. We can broaden query containment as follows.

Let $\varphi_1$ and $\varphi_2$ be conjunctive queries. A query $\varphi_1$ is said to be $p$-contained in $\varphi_2$, denoted $\varphi_1 \subseteq_p \varphi_2$, if and only if there exists a conjunctive query $\phi$, where $\varphi_1$ is equivalent to $\pi_L(\phi)$ ($\pi$ is relational projection), for some list $L$ of columns in head($\phi$) taken in the original order, such that for all databases $d$, $\phi(d) \subseteq \varphi_2(d)$. Note that p-containment is a generalization of query containment since $L$ can be the list of all columns in $\varphi_1$.

**Testing p-containment of conjunctive queries**

The classical notion of a containment mapping can be generalized to define p-containment mappings. A $p$-containment mapping from a conjunctive query $\varphi_2$ to a conjunctive query $\varphi_1$ is a mapping $\mu$, from variables of $\varphi_2$ to variables and constants of $\varphi_1$, such that

1. $\mu(\text{body}(\varphi_2)) \subseteq \text{body}(\varphi_1)$, and
2. for every variable $X$ in head($\varphi_1$) there is a variable $Y$ in head($\varphi_2$), such that $\mu(Y) = X$.

Consider the following example

$$\varphi_1 = Q_1(X) \leftarrow R(X,Y), S(Y,Y), T(Y,Z)$$
$$\varphi_2 = Q_2(A, B) \leftarrow R(A, B), S(B, C)$$

There is a p-containment mapping $\mu = \{A/X, B/Y, C/Y\}$ from $\varphi_2$ to $\varphi_1$.

We can now use p-containment mappings to test p-containment of conjunctive queries.

**Theorem 1** A query $\varphi_1$ is $p$-contained in a query $\varphi_2$ if and only if there is a $p$-containment mapping from $\varphi_2$ to $\varphi_1$.

**Proof.**

Let $\mu$ be a p-containment mapping from $\varphi_2$ to $\varphi_1$, and let $d$ be an arbitrary database. A fact $t_1$ in $\varphi_1(d)$ is generated by some valuation $\sigma$. Then $\sigma \circ \mu$ is a valuation that generates the corresponding fact $t_2$ in $\varphi_2(d)$. To see that this is indeed so, let $b \in \text{body}(\varphi_2)$. Then $\sigma \circ \mu(b) = \sigma(c) \in d$, for some $c \in \text{body}(\varphi_1)$.

Therefore, $\sigma(\phi(b)) \in d$. From requirement 2 of a p-containment mapping it follows that $\sigma(\phi(b)) \in d$. Thus, $\varphi_1 \subseteq_p \varphi_2$.

Let $\varphi_1 \subseteq_p \varphi_2$. Let $d$ be the canonical database that is the “frozen” body($\varphi_1$). By the definition of p-containment there exist a conjunctive query $\psi$, such that $\psi(d) \subseteq \varphi_2(d)$ and $\varphi_1 = \pi_L(\psi)$ for some ordered list $L$ of columns in head($\psi$). Obviously, $\varphi_1(d)$ contains a fact $t_1$, which is the “frozen” head($\varphi_1$). Since $\varphi_1 = \pi_L(\psi)$ there must be a fact $t_2$ in $\psi(d)$, such that $\pi_L(t_2) = t_1$. Since $\psi(d) \subseteq \varphi_2(d)$, we have $t_2 \in \varphi_2(d)$.

Let $\sigma$ be a valuation that generates the fact $t_2$ in $\varphi_2(d)$. Let $\rho$ be the the “freezing” mapping, which also is a valuation that generates the fact $t_1$ in $\varphi_1(d)$. Then $\rho^{-1} \circ \sigma$ is a p-containment mapping from $\varphi_2$ to $\varphi_1$.

To see that this is indeed so note two things. First, that each subgoal $b \in \text{body}(\varphi_2)$ is mapped by $\sigma$ to some fact in $d$, which is a frozen version of some subgoal $c \in \text{body}(\varphi_1)$, so $\rho^{-1} \circ \sigma$ maps $b$ to the unfrozen fact, that is to $c$ itself.

Second, note that those variables in head($\varphi_2$) that are also in head($\varphi_1$) are mapped by $\sigma$ to constants in the fact $t_1$, which is the frozen head($\varphi_1$), so that all of the head variables in $\varphi_1$ are covered. Thus $\rho^{-1} \circ \sigma$ maps corresponding variables in head($\varphi_2$) to the unfrozen head($\varphi_1$). Thus, $\rho^{-1} \circ \sigma$ is a p-containment mapping from $\varphi_2$ to $\varphi_1$. $\blacksquare$
3 Source Collections and Tableaux

Let \( \text{loc} \) be a countably infinite set \( \{V, V_1, V_2, \ldots\} \) of local relation names. The local relation names have arities, and atoms over local relation names are defined in the same way as atoms over relation names in \( \text{rel} \). To distinguish between relations (relation names) in \( \text{rel} \) and in \( \text{loc} \) we will henceforth call the former global relations (relation names).

A source \( S \) is a pair \((\varphi, v)\), where \( \varphi \) is a conjunctive query and \( v \) is a finite set of facts over head(\( \varphi \)).

A source collection \( S \) is a finite set of sources. The (global) schema of \( S \), denoted \( \text{sch}(S) \) is the set consisting of all the global relation names occurring in the bodies of the defining conjunctive queries of the sources in \( S \). The description of \( S \), denoted \( \text{desc}(S) \) is obtained from \( S \) by dropping the extension from every pair in \( S \). In other words, a source collection \( S \) has two “schemas,” \( \text{sch}(S) \) which is the “global world view” and \( \text{desc}(S) \), which describes the defining views. The extension of a source collection \( S \), denoted \( \text{ext}(S) \), is the union of all facts in sources in \( S \).

A source collection \( S \) defines a set of possible databases, denoted \( \text{poss}(S) \), as follows:

\[
\text{poss}(S) = \{d \in \text{sch}(S) : v_i \subseteq \varphi_i(d) \text{ for all sources } S_i = (\varphi_i, v_i) \text{ in } S\}.
\]

Note that \( \text{poss}(S) \) is infinite. We will now consider the problem of finitely representing an infinite set of databases. For this we invoke the venerable tableau.

Sets of databases and tableaux

In order to reason about tableaux we have to define a few concepts that are applicable to sets of global databases represented by tableaux.

Let \( \mathcal{X} \) and \( \mathcal{Y} \) be two enumerable sets of global databases over \( R \). We say that \( \mathcal{X} \) and \( \mathcal{Y} \) are coinital if they have the same \( \subseteq \)-minimal elements. Coinitiality is denoted \( \mathcal{X} \approx \mathcal{Y} \).

Let \( \Omega \) be the set of all queries expressible in a query language that we by abuse of notation also call \( \Omega \). Then \( \mathcal{X} \) and \( \mathcal{Y} \) are said to be \( \Omega \)-equivalent, denoted \( \mathcal{X} \equiv_\Omega \mathcal{Y} \), if for all queries \( Q \in \Omega \) we have

\[
\bigcap_{d \in \mathcal{X}} Q(d) = \bigcap_{d \in \mathcal{Y}} Q(d).
\]

The intuition behind \( \Omega \)-equivalence is that \( \mathcal{X} \) and \( \mathcal{Y} \) are indistinguishable as far as the certain information derivable by queries in \( \Omega \) are concerned. Thus, if a user can only pose queries in \( \Omega \), he cannot distinguish between \( \mathcal{X} \) and \( \mathcal{Y} \).

The following lemma is proved in the seminal paper [IL84].

**Lemma 1** Let \( \Omega \) be a monotone query language. If \( \mathcal{X} \approx \mathcal{Y} \), then \( \mathcal{X} \equiv_\Omega \mathcal{Y} \).

Of particular interest to us is of course choosing \( \Omega \) to be the set of all unions of conjunctive queries, which, it goes without saying, is a monotone query language.

We now define tableaux [Men84], which are intended to concisely and finitely represent a large or infinite set of possible instances.

Let \( R = \{R_1, R_2, \ldots, R_n\} \) be a set of relation names. A tableau \( T \) over \( R \) is a finite set of atoms over the \( R_i \)’s. Note that the same variable might appear in several atoms in \( T \).

A tableau \( T \) over schema \( R \) represents a set of databases over \( R \). This set is denoted \( \text{rep}(T) \), and it is defined by

\[
\text{rep}(T) = \{d : \text{there is a valuation } \sigma \text{ such that } \sigma(T) \subseteq d\}.
\]

The definition says that a database \( d \) is represented by a tableau \( T \), if there is a valuation \( \sigma \) such that when all variables in \( T \) are replaced by their image under \( \sigma \), the set of facts thus obtained is a subset of \( d \).

In order to compare tableaux, we need the concept of substitution. A substitution is a valuation, except that we allow variables to be mapped into variables, not only constants. Thus, a substitution \( \theta \)
is a function from (a subset of) \( \text{dom} \cup \text{var} \) to \( \text{dom} \cup \text{var} \), keeping in mind that constants have to be mapped to themselves.

Given two atoms \( t \) and \( u \) over the same relation name, we say that \( t \) is \emph{subsumed} by \( u \), denoted \( t \leq_{\text{sub}} u \), if there is a substitution \( \theta \), such that \( \theta(u) = t \). A tableau \( T \) is \emph{subsumed} by a tableau \( U \) denoted \( T \leq_{\text{sub}} U \), if for every atom \( t \in T \) there is an atom \( u \in U \), such that \( t \leq_{\text{sub}} u \). A tableau \( T \) is \emph{subsumption equivalent} to a tableau \( U \) denoted \( T \equiv_{\text{sub}} U \), if \( T \leq_{\text{sub}} U \) and \( U \leq_{\text{sub}} T \).

The following little lemma gives a semantic characterization of subsumption equivalence.

\textbf{Lemma 2} Let \( \Omega \) be a set of all projection queries. Then \( \text{rep}(T) \equiv_{\Omega} \text{rep}(U) \) if and only if \( T \equiv_{\text{sub}} U \). \( \blacksquare \)

In other words, if \( T \) and \( U \) are subsumption equivalent, then for any subset of the columns, \( T \) and \( U \) contain the same facts over these columns. Compared to letting \( \Omega \) be the set of all (unions of) conjunctive queries, subsumption equivalence cannot account for repeated variables. Such repeated variables might allow the user to infer more certain information using subsequent joins. We will return to this point in the concluding section.

\textbf{Representing poss}(\( S \)) by a tableau

Now the set \( \text{poss}(S) \) can be conveniently represented by a tableau over schema \( \text{sch}(S) \), denoted \( T(S) \), such that \( \text{rep}(T) = \text{poss}(S) \). To construct \( T \) we shall follow the approach in [GM99]. We define a function, which we by abuse of notation also denote \( T \), from sources with defining view \( \varphi \), where the body of \( \varphi \) consists of atoms over relation names in \( R \), to tableau over \( R \). We also need an auxiliary function \( \text{refresh} \), that, when applied to a set of atoms, replaces all variables with fresh ones. Given a source \( S = (\varphi, v) \), we set

\[
T(S) = \bigcup_{u \in v} \{ \text{refresh}(\sigma(\text{body}(\varphi))) : \sigma(\text{head}(\varphi)) = u \text{ for some valuation } \sigma \}.
\]

For example, if \( S = \{ V(X, Z) \rightarrow R(X, Y), S(Y, Z), \{ V(a, b), V(c, d) \} \} \), then \( T(S) = \{ R(a, Y_1), S(Y_1, b), R(c, Y_2), S(Y_2, d) \} \), where \( Y_1 \) and \( Y_2 \) are fresh variables. When there are several sources in \( S \) we set

\[
T(S) = \bigcup_{S \in S} T(S).
\]

The tableau constructed by the function \( T \) has the following desirable property.

\textbf{Theorem 2} \( \text{rep}(T(S)) = \text{poss}(S) \).

\textbf{Proof}. Let \( d \in \text{rep}(T(S)) \). To prove that \( d \in \text{poss}(S) \) we need to show that for all sources \( S_i = (\varphi_i, v_i) \) in \( S \), we have \( v_i \subseteq \varphi_i(d) \). Since \( d \in \text{rep}(T(S)) \) there is a valuation \( \sigma \) such that \( \sigma(T(S)) \subseteq d \). Let \( S_i = (\varphi_i, v_i) \) be an arbitrary source in \( S \), and let \( t \) be an arbitrary fact in \( v_i \). Then there must be a substitution \( \theta \), such that \( t = \theta(\text{head}(\varphi_i)) \) and all facts in \( \theta(\text{body}(\varphi_i)) \) are in \( T(S) \). It follows that \( \theta(\sigma(\text{body}(\varphi_i))) \subseteq d \) and, consequently, \( \theta(\sigma(\text{head}(\varphi_i))) \in d \). Since \( \theta(\sigma(\text{head}(\varphi_i))) = t \), we have \( v_i \subseteq \varphi_i(d) \) as desired.

For inclusion in the other direction, let \( d \in \text{poss}(S) \). From construction of \( T(S) \) it immediately follows that there is a valuation \( \sigma \) such that \( \sigma(T(S)) \subseteq d \) and, thus, that \( d \in \text{rep}(T(S)) \). \( \blacksquare \)

\section{Querying Source Collections}

Let \( S \) be source collection, and \( \varphi \) a conjunctive query, such that the body of \( \varphi \) consists of atoms over relation names in \( \text{sch}(S) \). Now \( \varphi \) applied to \( S \) defines the \emph{exact answer}:

\[
\varphi(S) = \{ \varphi(d) : d \in \text{poss}(S) \}.
\]

The definition essentially says that since the source collection corresponds to a set of databases, the answer should also correspond to a set, obtained by evaluating the query pointwise.
Computing the exact answer from the tableau

Now since we are able to construct a database template \( T \) representing all databases in \( \text{pos}(S) \) it is natural to extend the standard query evaluation mechanism to operate on database templates.

Let \( T \) be a tableau over \( R \). Given a conjunctive query \( \varphi \) over \( R \), our evaluation \( \hat{\varphi} \) (which is basically the “naive evaluation” of [IL84]) is as follows. Recall that a substitution is a valuation, except that variables can be mapped into variables, not only constants. Then

\[
\hat{\varphi}(T) = \{ \theta(\text{head}(\varphi)) : \theta(\text{body}(\varphi)) \subseteq T \text{ for some substitution } \theta \}.
\]

For example, let \( \varphi = Q(X, Y, Z) \leftarrow R(X, Y), S(Y, Z) \) and \( T = \{ R(a, b), R(d, X), S(b, c), S(X, e), S(Y, f) \} \). Then \( \hat{\varphi}(T) = \{ Q(a, b, c), Q(d, X, e) \} \).

Clearly, if our definition of \( \hat{\varphi} \) is semantically meaningful, then we should expect that it approximates the information given by \( \varphi(S) \) in some natural sense. Indeed, our extended semantics has the following property:

**Theorem 3** \( \text{rep}(\hat{\varphi}(T)) \approx \varphi(\text{rep}(T)) \).

**Proof.** Let \( d \) be a \( \subseteq \)-minimal element in \( \text{rep}(\hat{\varphi}(T)) \). Then there exist a valuation \( \sigma \) such that \( \sigma(\hat{\varphi}(T)) = d \). Let \( t \) be an arbitrary fact in \( d \). Then there is a fact \( u \in \hat{\varphi}(T) \) such that \( \sigma(u) = t \), and there is a substitution \( \theta \) such that \( \theta(\text{body}(\varphi)) \subseteq T \) and \( u = \theta(\text{head}(\varphi)) \).

Let \( \sigma' \) be an extension of \( \sigma \) that maps every variable that is in \( T \) but not in \( \hat{\varphi}(T) \) to a distinct new constant. Since \( \theta(\text{body}(\varphi)) \subseteq T \), we have \( \sigma'\theta(\text{body}(\varphi)) \subseteq \sigma'(T) \). It now follows that \( t = \sigma'(\theta(\text{head}(\varphi))) \in \varphi(\sigma'(T)) \). Note that \( \sigma'(T) \) is a \( \subseteq \)-minimal element in \( \text{rep}(T) \). From the monotonicity of \( \varphi \) it follows that \( \varphi(\sigma'(T)) \) is a \( \subseteq \)-minimal element in \( \varphi(\text{rep}(T)) \). We have established that \( d \subseteq \varphi(\sigma'(T)) \).

That concludes the proof that any \( \subseteq \)-minimal element \( d \) in \( \text{rep}(\hat{\varphi}(T)) \) is in \( \varphi(\text{rep}(T)) \).

For inclusion in the other direction let \( d \) be a \( \subseteq \)-minimal element in \( \varphi(\text{rep}(T)) \). Then there is a valuation \( \sigma \), such that \( d = \varphi(\sigma(T)) \). Let \( t \) be an arbitrary tuple in \( d \). Then there is a valuation \( \rho \), such that \( t = \rho(\text{head}(\varphi)) \) and all facts in \( \rho(\text{body}(\varphi)) \) are in \( \sigma(T) \). Now we have two cases to consider.

**Case 1:** The valuation \( \sigma \) is one-to-one. Then there is an inverse \( \sigma^{-1} \), and hence \( \sigma^{-1}(\rho(\text{body}(\varphi))) \subseteq \sigma^{-1}(\sigma(T)) = T \), and consequently \( \sigma^{-1}(t) = \sigma^{-1}(\rho(\text{head}(\varphi))) \) is in \( \hat{\varphi}(T) \). Since \( \sigma(\hat{\varphi}(T)) \in \text{rep}(\hat{\varphi}(T)) \), it follows that \( t \in \sigma(\hat{\varphi}(T)) \in \text{rep}(\hat{\varphi}(T)) \). Likewise, if \( t' \) is any other tuple in \( d = \varphi(\sigma(T)) \), it is generated by some valuation \( \rho' \), and we have \( \sigma^{-1}(t') = \sigma^{-1}(\rho'(\text{head}(\varphi))) \in \hat{\varphi}(T) \). Therefore \( d \subseteq \sigma(\hat{\varphi}(T)) \).

**Case 2:** There is (at least one) pair of distinct variables \( X \) and \( Y \) in \( T \), such that \( \sigma(X) = \sigma(Y) \). If \( \sigma(X) = \rho(U) \), and \( \sigma(Y) = \rho(W) \), for \( U \neq W \), then the valuation \( \omega \), that is like \( \sigma^{-1} \circ \rho \), except \( \omega(U) = X \), and \( \omega(V) = Y \), gives us \( \omega(\text{body}(\varphi)) \subseteq T \), and \( \sigma^{-1}(t) = \omega(\text{head}(\varphi)) \in \hat{\varphi}(T) \). Consequently \( t = \sigma(\sigma^{-1}(t)) \in \sigma(\hat{\varphi}(T)) \).

Suppose then that \( \sigma(X) = \sigma(Y) = \rho(W) \), and that there are (at least) two occurrences of \( W \) in \( \text{body}(\varphi) \). Consider now the valuation \( \sigma' \), that is exactly like \( \sigma \), except it maps \( Y \) to a fresh constant, say \( a \). Clearly \( t \notin \varphi(\sigma'(T)) \), and any fact in \( \varphi(\sigma'(T)) \) is also in \( \varphi(\sigma(T)) \) (because there is an embedding of \( \sigma'(T) \) into \( \sigma(T) \).) Therefore we have a contradiction to the assumption that \( t \) belonged to a \( \subseteq \)-minimal element of \( \varphi(\text{rep}(T)) \).

As a consequence we now have a method for computing an \( \approx \)-approximation of \( \varphi(S) \).

**Corollary 1** \( \text{rep}(\hat{\varphi}(T(S))) \approx \varphi(S) \).

In other words, first invert the source extensions through their definitions, then apply the \( \hat{\varphi} \)-evaluation of the user query \( \varphi \) on the resulting tableau. The result of the evaluation is another tableau, which the user perceives as a relation with nulls.

The problem of computing exact answer to a user query was not addressed in the literature except for a brief discussion in [GM99], instead all of the algorithms are aimed at computing the possible or, most commonly, certain answer.
The possible answer can be defined as \( \varphi^*(S) = \bigcup \{ \varphi(d) : d \in \text{poss}(S) \} \). The certain answer can be defined as \( \varphi^*(S) = \bigcap \{ \varphi(d) : d \in \text{poss}(S) \} \). Given the exact answer that is obviously most informative of all answers, we can obtain the possible answer and the exact answer as follows.

**Lemma 3** \( \varphi_*(S) = \cap \text{rep}(\tilde{T}(S))), \) and \( \varphi^*(S) = \cup \text{rep}(\tilde{T}(S))). \)

However, computing \( \tilde{T}(S) \) might involve a lot of redundant work, since it amounts to constructing the tableau corresponding to the entire \( \text{ext}(S) \), whereas the local relations that are in \( \text{body}(\varphi) \) might be mentioned in only few source definitions. Furthermore, the query might have selections and joins that could be computed directly at the sources.

### Computing the exact answer directly on the source collection

In view-centric information integration systems a query processor is in charge of reformulating queries written in terms of this global schema to queries on the appropriate sources. This process is also known as query rewriting. We can extend the notion of rewriting to \( p \)-rewriting.

To this end we need a few concepts. The *expansion* of a query \( \varphi \) over \( \text{desc}(S) \), denoted \( \varphi^{exp} \), is obtained from \( \varphi \) by replacing all the sources in \( \varphi \) with their definitions. Existential variables in a source definition are replaced by fresh variables in \( \varphi^{exp} \).

Let \( S \) be a source collection and \( \varphi \) be a conjunctive query over \( \text{desc}(S) \). The query \( \psi \) is a \( p \)-contained rewriting of \( \varphi \) using \( S \) if \( \psi^{exp} \subseteq_p \varphi \). Let \( \psi \) be a \( p \)-contained rewriting of \( \varphi \). We define the \( \varphi \)-evaluation of \( \psi \), denoted \( \psi_\varphi \), as follows

\[
\psi_\varphi(S) = \{ \sigma_\mu(\text{head}(\varphi)) : \sigma(\text{body}(\psi)) \subseteq \text{ext}(S) \},
\]

where \( \mu \) is a \( p \)- containment mapping from \( \varphi \) to \( \psi^{exp} \), \( \sigma \) is a valuation we extend to \( \sigma_\mu \) by setting

\[
\sigma_\mu(X) = \begin{cases} 
\sigma(\mu(X)), & \text{if } \mu(X) \text{ occurs in } \text{head}(\psi) \\
\text{a fresh variable}, & \text{otherwise}
\end{cases}
\]

Note that it is possible that there is more than one \( p \)-containment mapping from \( \varphi \) to \( \psi^{exp} \). However, as formalized in the following Lemma, choosing one mapping over the other does not affect \( \psi_\varphi(S) \).

**Lemma 4** Let \( \varphi \), and \( \phi \) be conjunctive queries, such that \( \phi^{exp} \subseteq \varphi \), and let \( \mu_1 \) and \( \mu_2 \) be containment mappings from \( \varphi \) to \( \phi^{exp} \). Then for all source collections \( S \), \( \{ \sigma_{\mu_1}(\text{head} \varphi) : \sigma_{\mu_1}(\varphi) \subseteq \text{ext}(S) \} = \{ \sigma_{\mu_2}(\text{head} \varphi) : \sigma_{\mu_2}(\varphi) \subseteq \text{ext}(S) \} \), up to renaming of the fresh variables.

Now we can define \( \tilde{\varphi}(S) \) as

\[
\tilde{\varphi}(S) = \bigcup \{ \psi_{\varphi}(S) : \psi^{exp} \subseteq_p \varphi \},
\]

and state the following important result:

**Theorem 4** \( \tilde{\varphi}(S) \equiv_{sub} \tilde{T}(S)). \)

**Proof.** Let \( t \) be an arbitrary atom in \( \tilde{\varphi}(S) \). Then there was conjunctive query \( \psi \) (over \( \text{desc}(S) \)) in the union of maximally-contained \( p \)-rewritings of \( \varphi \) and a \( \varphi \)-evaluation of \( \psi \), using a valuation \( \sigma_\mu \) such that \( t = \sigma_\mu(\text{head} \varphi) \), and \( \sigma(\text{body}(\psi)) \subseteq \text{ext}(S) \), where \( \mu \) is a containment mapping from \( \varphi \) to \( \psi^{exp} \).

Since \( \sigma(\text{body}(\psi)) \subseteq \text{ext}(S) \), it means that all atoms in \( \sigma(\text{body}(\psi^{exp})) \) are in \( \text{T}(S) \) (with fresh existential variables). Since \( \mu \) is a containment mapping from \( \varphi \) to \( \psi^{exp} \), we have that \( \sigma(\mu(\text{body}(\varphi))) \subseteq \text{ext}(S) \). Thus \( \sigma(\mu(\text{head}(\varphi))) \in \tilde{T}(S) \). Now \( \sigma(\mu(\text{head}(\varphi))) \) is equal to \( \sigma_\mu(\text{head}(\varphi)) \), except for positions that don’t occur in \( \text{head}(\psi) \), these have been replaced by fresh variables in \( \sigma_\mu(\text{head}(\varphi)) \). If we now define a substitution \( \theta \) that maps each of these fresh variables to the variable or constant in the corresponding position in \( \sigma(\mu(\text{head}(\varphi))) \), we get that \( \theta(\sigma(\mu(\text{head}(\varphi)))) = \sigma_\mu(\text{head}(\varphi)) \), and consequently \( \sigma_\mu(\text{head}(\varphi)) \leq_{sub} \sigma(\mu(\text{head}(\varphi))) \). This means that \( \tilde{\varphi}(S) \leq_{sub} \tilde{T}(S) \).
For the proof of inclusion in the other direction, let \( t \) be an arbitrary atom in \( \hat{\varphi}(T(S)) \). Suppose \( \text{body}(\varphi) \) consists of atoms \( b_1, b_2, \ldots, b_n \). Then there is a substitution \( \theta \), such that \( \theta(b_i) \in T(S) \), for all \( i \in \{1, \ldots, n\} \). But each atom \( \theta(b_i) \) is in \( T(S) \) because there is a source \( S_{t_i} = (\varphi_i, v_{t_i}) \), a valuation \( \sigma_i \), and a fact \( t_i \in v_{t_i} \), such that \( t_i = \sigma_i(\text{head}(\varphi_i)) \) and \( \theta(b_i) \in \text{refresh}(\sigma_i(\text{body}(\varphi_i))) \). If we take \( \psi \) to be the query with \( \text{body}(\psi) = \sigma_1(\text{head}(\varphi_1)), \ldots, \sigma_n(\text{head}(\varphi_n)) \), and \( \text{head}(\psi) = (\sigma_1 \cup \sigma_2 \cup \cdots \cup \sigma_n)(\text{head}(\varphi)) \), we have a containment mapping (namely \( \theta \)), from \( \varphi \) to \( \psi^{\text{exp}} \). Consequently \( \psi \) will be an element in the union of queries \( \hat{\varphi} \), and obviously \( \psi \) generates the fact \( t \) when applied to \( S \). Since \( t \leq_{\text{sub}} t \), we have established that \( \hat{\varphi}(S) \leq_{\text{sub}} \hat{\varphi}(T) \).

In the next section we give an algorithm that, for a given conjunctive query \( \varphi \), computes a finite union of conjunctive queries equivalent to \( \hat{\varphi} \).

## 5 The P-bucket Algorithm

Since we generalized the notion of containment mapping to p-containment mapping it is only natural that any rewriting algorithm, which is based on containment mappings, can be extended to produce p-rewriting.

The following algorithm is a straightforward modification of bucket algorithm [LRO96] and, therefore, we call it p-bucket algorithm. Given a query \( \varphi \) the p-bucket algorithm proceeds in two steps. In the first step, the algorithm creates a bucket for each subgoal in \( \varphi \). Then the buckets are populated by source atoms (subgoals) that are relevant to answering the particular subgoal. More specifically, consider a bucket for a subgoal \( b_\varphi \) of \( \varphi \), and a source \( S_i = (\varphi_i, v_i) \). If \( \text{body}(\varphi) \) contains a subgoal \( b_{\varphi_i} \) such that there is a (most general) unifier \( \theta \) for \( b_\varphi \) and \( b_{\varphi_i} \), then \( \theta(\text{head}(\varphi_i)) \) is put in the bucket of subgoal \( b_\varphi \). In case the subgoal \( b_\varphi \) unifies with more than one subgoal in a source \( S_i \), the bucket of \( b_\varphi \) will contain multiple occurrences of unified \( \text{head}(\varphi_i) \).

In the second step, the algorithm considers query rewritings that are conjunctive queries, each consisting of one conjunct from every bucket. The head of each rewriting is the projection of variables that are in the body of this rewriting, if the projection results in an empty list the rewriting is discarded. For each rewriting, the algorithm checks whether it’s expansion is p-contained in the query. If so, the rewriting is added to the answer. Though not required, a check can be added to determine that the resulting rewriting is not redundant. Hence, the result of algorithm is a union of conjunctive rewritings.

**Theorem 5** The union of all rewritings produced by the p-bucket algorithm relative to a query \( \varphi \) is equivalent to the union of all p-contained rewritings of \( \varphi \).

**Proof.** (Sketch.) The p-bucket algorithm produces only semantically correct rewritings since it tests for p-containment of each of candidate solutions.

For the proof that the output of the algorithm contains all semantically correct rewritings we have to prove that if there exists a p-rewriting \( \psi \) of a given conjunctive query \( \varphi \) then there will be a p-rewriting \( \chi \) in the output of the p-bucket algorithm, such that \( \psi \leq_p \chi \).

It has been known since Chandra and Merlin’s paper [CM77] that every conjunctive query has a unique (up to renaming of variables) minimal equivalent query that can be obtained by deletion of zero or more atoms. Let us call the minimal equivalent of \( \psi \) \( \psi_{\text{min}}^{\text{exp}} \) and the minimal equivalent of \( \chi \) \( \chi_{\text{min}}^{\text{exp}} \). Since \( \psi \) is a p-rewriting of \( \varphi \) \( \psi^{\text{exp}} \leq_p \varphi \) and consequently \( \psi_{\text{min}}^{\text{exp}} \leq_p \varphi \). It is easy to see that \( \psi_{\text{min}} \) cannot have more subgoals than \( \varphi \) because each subgoal of \( \psi_{\text{min}} \) covers at least one subgoal of \( \varphi \). We now have two cases: either \( \psi_{\text{min}} \) has the same number of subgoals as \( \varphi \), or \( \psi_{\text{min}} \) has fewer subgoals then \( \varphi \).

If \( \psi_{\text{min}} \) has the same number of subgoals as \( \varphi \) then each subgoal of \( \psi_{\text{min}} \) would be placed in the corresponding bucket by the first phase of p-bucket algorithm. In the second phase the algorithm produces cross-product of the contents of all buckets and, thus, it would produce \( \chi \) that has all of \( \psi_{\text{min}} \).
subgoals. Then the algorithm would compute the \( \text{head}(\chi) \) that would be the longest possible list of variables of the \( \text{head}(\varphi) \). Therefore, the p-bucket algorithm would produce \( \chi \) such that \( \psi_{\text{min}} \subseteq_p \chi \).

If \( \psi_{\text{min}} \) has fewer subgoals then \( \varphi \) then each subgoal of \( \psi_{\text{min}} \) would be placed in the corresponding bucket by the first phase of p-bucket algorithm. In the second phase the algorithm produces cross-product of the contents of all buckets and, thus, it would produce \( \chi \) that has all of \( \psi_{\text{min}} \) subgoals plus some redundant subgoals. Then the algorithm would compute the \( \text{head}(\chi) \) that would be the longest possible list of variables of the \( \text{head}(\varphi) \). Therefore, the p-bucket algorithm would produce \( \chi \) such that \( \psi_{\text{min}} \subseteq_p \chi_{\text{min}} \).

The equivalence of the claim of the theorem now follows from the characterization of equivalence of unions of conjunctive queries given in [SY80].

We illustrate the algorithm with the example given in the introduction. The first step of the algorithm will construct and populate two buckets, one for each of the subgoals in the query:

\[
\begin{array}{c}
S_1(\text{Pname, Email}) \\
S_2(\text{Pname, Office}) \\
S_3(\text{Pname, Area}) \\
S_4(\text{Pname, Dname})
\end{array}
\]

The second step of the algorithm produces the following p-rewritings:

\[
Q_1(\text{Pname, Email}) \leftarrow S_1(\text{Pname, Email}), S_4(\text{Pname, compsci})
\]
\[
Q_2(\text{Pname, Office}) \leftarrow S_2(\text{Pname, Office}), S_4(\text{Pname, compsci})
\]
\[
Q_3(\text{Pname, Area}) \leftarrow S_3(\text{Pname, Area}), S_4(\text{Prof, compsci})
\]

Note that the second step of the algorithm can be easily modified to insert the fresh variables representing unknown values in the head of the rewritings. The result of the algorithm can then be presented to the user as a table with nulls as in the example in the introduction.

6 Summary

For an illustrative example, let \( S \) consist of a single source with definition \( V(D, A, C) \leftarrow R(D, A, B) \), \( S(B, C) \) and extension \( \{V(c, a, a), V(d, a, b)\} \).

- \( \text{poss}(S) \) is equal to the closure under superset of \( \{R(c, a, a), S(a_0, a), R(c, a, a_0), S(a_0, a)\} \), \( \{R(c, a, a_0), S(a_0, a), R(c, a, a_1), S(a_1, a), \ldots\} \), where \( a_0, a_1, \ldots \) is an enumeration of \( \text{dom} \).
- \( T(S) = \{R(c, a, X), S(X, a), R(d, a, Y), S(Y, b)\} \). It is easy to see that \( \text{rep}(T(S)) = \text{poss}(S) \).

Consider now \( \varphi = Q(W, X, Z) \leftarrow R(W, X, Y), S(Z, X) \).

- We have \( \varphi(S) \approx_x \varphi(T(S)) = \{Q(c, a, X), Q(d, a, X)\} \). Let’s call this tableau \( T \).
- The p-bucket algorithm will produce the rewriting \( \psi = Q(W, X, E) \leftarrow V(W, X, Y) \), where \( E \) is a fresh variable. Consequently, \( \varphi(S) \equiv_{\text{sub}} \varphi(S) = \{Q(c, a, X), Q(d, a, Y)\} \). We call this tableau \( U \).
- Note that \( T \equiv_{\text{sub}} U \), but \( T \not\equiv_\Omega U \) when \( \Omega \) is equal to all (unions of) conjunctive queries. Although \( T \) and \( U \) have the same partial facts, \( T \) contains the additional information that the two variables represent the same unknown value. This allows the user to extract more certain information from \( T \) than from \( U \). For example, let \( \psi = P(U, W) \leftarrow Q(U, Y, X), Q(W, Z, X) \). Then the certain answer to \( \psi(\text{rep}(T)) \) is \( \{P(c, d)\} \), and the certain answer to \( \psi(\text{rep}(U)) \) is empty.
Thus, unless the user wants to materialize the exact answer and obtain certain answers for subsequent queries on it, we conclude that for practical purposes in information integration systems, the tableau $\tilde{\varphi}(S)$ is a sufficient approximation of $\varphi(S)$.

Note that in the introductory example the tableau representing the inversion of sources $S_1, S_2, S_3,$ and $S_4$ will never contain repeated occurrences of any variable. In this example the $\tilde{\varphi}$ and $\hat{\varphi}$ evaluations of the user query $\varphi = Q(Pname, Email, Office, Area) \leftarrow Prof(Pname, Email, Office, Area), Dept(Pname, compsci)$ both produce the same tableau.

References


Learning Efficient Value Predictors for Speculative Plan Execution

Greg Barish and Craig A. Knoblock
University of Southern California / Information Sciences Institute
4676 Admiral Way, Marina del Rey, CA 90292
{barish, knoblock}@isi.edu

Abstract
Speculative plan execution can be used to significantly improve the performance of information gathering plans. However, its impact is closely tied to the ability to predict data values at runtime. While caching can be used to issue future predictions, such an approach often scales poorly with large data sources and is unable to make intelligent predictions about novel hints, even when there is an obvious relationship between the hint and the predicted value. In this paper, we describe how learning decision trees and transducers can lead to a more efficient value prediction system as well as one capable of making intelligent predictions about new hints. Our initial results validate these claims in the context of the speculative execution of one common type of information gathering plan.

Introduction
Improving the performance of network-bound information gathering plans remains an ongoing research challenge. Recent systems that execute such plans (Ives et al. 1999; Hellerstein et al. 2000; Naughton et al. 2001) employ adaptive query processing techniques to improve execution efficiency. Adaptive query processing is unique to network-bound information gathering; it addresses the unpredictability of source response time and the potentially sizeable results returned by responding to runtime events or by dynamically re-ordering tuples or operators.

Speculative plan execution (Barish and Knoblock 2002) is a new technique that can be used to dramatically improve the efficiency of network-bound information gathering plans. The idea involves using data seen early in plan execution as a basis for issuing predictions about data needed during later parts of execution. This allows sequential data dependency chains to be broken and parallelized, leading to better average performance.

To maximize the impact of speculative execution on plan performance, a good value prediction strategy is required. The basic problem involves being able to use some hint \( h \) as the basis for issuing a predicted value \( v \). Of course, a few simple strategies can be applied. Caching is an obvious choice; we can note that particular hint \( h \) corresponds to a particular value \( v \), so that future receipt of \( h \) can lead to a prediction of \( v \).

However, there are two problems with caching. One is space-efficiency; prediction requires the storage of prior hint/value mappings. If the values being predicted are associated with large data sources, the resulting lookup table can become prohibitively large. The second problem is accuracy; caching only allows prediction of previously seen values. Thus, predictions cannot be issued for new hints, even when the former is a simple function of the latter or based on a subset of hint attributes.

In this paper, we propose a new approach that uses machine learning techniques to craft a prediction system that is both efficient and accurate. Our approach uses two learning mechanisms for issuing intelligent predictions: (a) decision trees, which classify hints based on their most informative attributes and (b) subsequential transducers, which translate hints into predicted values. Specifically, this paper contributes the following:

- An approach using decision trees as a means for classifying a hint into a prediction.
- An approach using subsequential transducers as a means for translating a hint into a prediction.
- An algorithm that unifies the incremental learning of both structures, based on the type of relationship observed between hint and predicted value.
- Initial results of the application of this algorithm to one common type of information gathering plan.

It is important to note that this work focuses on the challenge of what to speculate. Although we review it here briefly, prior work (Barish and Knoblock 2002) has addressed the details of one method for how to speculate.

The rest of this paper is organized as follows. The next section reviews information gathering and speculative execution. In Section 3, we describe how decision trees and transducers can be used to build efficient and intelligent predictors. Section 4 presents an algorithm that unifies the learning of both types of predictors. Finally, we present the initial results of applying our approach to the execution of a common type of information gathering plan.

Preliminaries
Information gathering plans retrieve, combine, and manipulate data located in remote sources. Such plans consist of a partially-ordered graph of operators \( O \), connected in producer/consumer fashion. Each operator \( O \) consumes a set of inputs \( \alpha \), fetches data or performs a computation based on that input, and produces one or more outputs \( \beta \). The types of operators used in information gathering plans varies, but most either retrieve or perform computations on data.
Operators process and transmit data in terms of relations. Each relation \( R \) consists of a set of attributes (i.e., columns) \( a_1, a_2 \), and a set of zero or more tuples (i.e., rows) \( t_1, t_2 \), each tuple \( t \) containing values \( v_{1}, v_{2} \). We can express relations with attributes and a set of tuples as:

\[
R(a_1, a_2) = \{(v_{11}, v_{21}), (v_{21}, v_{22}), \ldots, (v_{n1}, v_{n2})\}
\]

**Example Plan.** To illustrate, consider the plan executed by an information agent called *RepInfo*. This plan, shown in Figure 1, returns information about U.S. congressional officials. Given any U.S. postal address, *RepInfo* gathers the names, funding charts, and recent news related to U.S. federal officials (members of the Senate or House of Representatives) for that address. *RepInfo* retrieves this information via the following web data sources:

- **Vote-Smart**, to identify the officials for an address.
- **OpenSecrets**, for funding data about each official.
- **Yahoo News**, for recent news about each official.

In Figure 1, *Wrapper* operators retrieve data from Web sources, a *Select* operator filters federal officials from other types of officials, and a *Join* operator combines the funding and news data in order to return the entire result as a single output. *Wrapper*-style operators are very common in web information gathering plans. They work by extracting semi-structured data from a set of remote web pages into a relation of structured data.

![Figure 1: The RepInfo plan](image)

At the start of execution, an input postal address is used to query the Vote-Smart source, returning the set of federal officials for that location. A subsequent *Select* operator filters Senate and House officials. This subset is then used to query Yahoo News and OpenSecrets. Note that the latter requires additional retrieval steps in order to navigate to the page containing the funding data. The results of both are then joined together, providing the result shown in Figure 2.

The plan shown in Figure 1 is one common type of information gathering plan. Similar plans that combine data from two or more distinct sources can be found throughout prior research (Friedman and Weld 1997; Ives et al. 1999).

### Speculative Plan Execution

Normally, execution of an information gathering plan requires that each of its independent data flows be executed serially. For example, in Figure 1, there are two flows. The first one, \( f_1 \), includes the Vote-Smart, Select, Yahoo News, and Join operators. The second one, \( f_2 \), includes the Vote-Smart, Select, OpenSecrets, and Join operators. Per Amdahl’s Law, the time required to execute the plan will be no less than the slowest of these two.

Table 1 shows the average execution times for \( f_1 \) and \( f_2 \).

<table>
<thead>
<tr>
<th>Flow</th>
<th>Path</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f_1 )</td>
<td>VoteSmart→Select→Yahoo→Join</td>
<td>3280</td>
</tr>
<tr>
<td>( f_2 )</td>
<td>VoteSmart→Select→OpenSecrets→Join</td>
<td>6500</td>
</tr>
</tbody>
</table>

**Table 1: Execution time of RepInfo data flows**

Thus, execution time is \( \text{MAX}(3280, 6500) = 6500\) ms.

Speculative execution overcomes Amdahl’s Law by parallelizing a sequence of dependent operators through *predictions* about those dependencies. Predictions are made based on *hints* which occur earlier during the execution of a flow. For example, in Figure 1, we could use the input to Vote-Smart (the postal address) as a hint for predicting the likely set of representatives that will be communicated to the OpenSecrets and Yahoo operators. Thus, the Vote-Smart operator will be executing in parallel with the OpenSecrets and Yahoo wrappers, generally leading to faster execution. Of course, we need to ensure correctness by “guarding” against the release of speculative results until earlier predictions are confirmed.

Figure 3 shows the plan in Figure 1 modified for speculative execution. In the figure, two types operators have been added. The first is Speculate (labeled SPEC), which receives hints and answers, makes predictions, and issues confirmations. These inputs and outputs have been labeled for the first Speculate operator in the figure. The second new operator is SpecGuard (labeled GUARD), which prevents speculative results from exiting the plan until the predictions that led to those results has been confirmed. As the figure shows, speculation can be cascading (i.e., based on earlier speculation). Correctness is enforced by SpecGuard – only confirmed speculation can exit the plan (and affect the external world).

In summary, Figure 3 shows three instances of speculation: (a) the prediction of federal representatives based on street address, (b) the prediction of OpenSecrets member page URL based on the name of the official, and (c) the prediction of the OpenSecrets funding page URL based on the member page URL. In (Barish and Knoblock 2002), we demonstrate how executing this plan can result in an optimistic speedup of 3.65.

![Figure 3: RepInfo modified for speculative execution](image)
Data Value Prediction

While speculative plan execution can lead to substantial speedups, its effectiveness is directly related to the accuracy of the predictions issued. Good predictions improve the average execution time; poor predictions have the opposite effect. Although prediction based on caching prior hint/answer tuples is possible, this method is space-inefficient and unable to make predictions for new hints.

To build a more efficient and intelligent predictor, we show how machine learning techniques can be applied to the problem. In this section, we describe how two such techniques, decision trees and transducers, can be employed. Both are more efficient than caching and both allow predictions to be made about novel hints.

Prediction Using Decision Trees

Decision trees are an effective tool for classifying data. By calculating the information gain from a series of multi-attribute examples, we can use decision trees to identify and rank the attributes most useful when making a prediction. This last feature enables decision trees to issue reasonable predictions about novel hints.

For example, for the modified plan in Figure 3, we can use a subset of address attributes to predict the federal officials that will be fetched. Specifically, we are interested in predicting the two Senate members and single House of Representatives member for that address.

Intuitively, we know that a given senator can be associated with millions of addresses and a given representative can be associated with thousands. Decision trees can be used to help us identify that the state attribute of the address hint is the most informative for senators while city and zip code are the most informative hints when predicting representatives.

As a detailed example, let us consider building decision trees that classify House of Representatives members based on postal addresses. Table 2 shows 10 sample addresses and their corresponding representatives. Using the ID3-based C5.0 classifier based on (Quinlan 1986), the following decision list is constructed:

city = Culver City: Jane Harman (0)
city = Marina del Rey: Jane Harman (2)
city = Venice: Jane Harman (3)
city = Santa Monica: Henry Waxman (1)
city = Los Angeles: ...; zip <= 90064; Henry Waxman (1)
zip > 90064: Diane Watson (2)

We can then use this list to issue predictions for recurring and new hints. For example, we can predict Jane Harman for the recurring address 4676 Admiralty Way, Marina del Rey, CA as well as for the new address 4680 Admiralty Way, Marina del Rey, CA.

Note that predictions will not always be correct – for example, the above tree incorrectly predicts Jane Harman (not Diane E. Watson) as the representative for 4065 Glencoe Avenue, Marina del Rey, CA, 90292. However, since these are predictions for execution and not final answers, such inaccuracy is not fatal – a speculative execution system that ensures correctness can recover from such errors.

<table>
<thead>
<tr>
<th>Street</th>
<th>City</th>
<th>State</th>
<th>Zip</th>
<th>Representative</th>
</tr>
</thead>
<tbody>
<tr>
<td>14044 Panay Way</td>
<td>Marina del Rey</td>
<td>CA</td>
<td>90292</td>
<td>Jane Harman</td>
</tr>
<tr>
<td>4676 Admiralty Way</td>
<td>Marina del Rey</td>
<td>CA</td>
<td>90292</td>
<td>Jane Harman</td>
</tr>
<tr>
<td>101 Washington Blvd</td>
<td>Venice</td>
<td>CA</td>
<td>90292</td>
<td>Jane Harman</td>
</tr>
<tr>
<td>1301 Main St</td>
<td>Venice</td>
<td>CA</td>
<td>90291</td>
<td>Jane Harman</td>
</tr>
<tr>
<td>1906 Lincoln Blvd</td>
<td>Venice</td>
<td>CA</td>
<td>90291</td>
<td>Jane Harman</td>
</tr>
<tr>
<td>2107 Lincoln Blvd</td>
<td>Santa Monica</td>
<td>CA</td>
<td>90405</td>
<td>Henry Waxman</td>
</tr>
<tr>
<td>2222 S Centellas Ave</td>
<td>Los Angeles</td>
<td>CA</td>
<td>90064</td>
<td>Henry Waxman</td>
</tr>
<tr>
<td>4056 Glencoe Ave</td>
<td>Marina del Rey</td>
<td>CA</td>
<td>90290</td>
<td>Diane Watson</td>
</tr>
<tr>
<td>3970 Berryman Ave</td>
<td>Los Angeles</td>
<td>CA</td>
<td>90066</td>
<td>Diane Watson</td>
</tr>
<tr>
<td>31481 Washington Blvd</td>
<td>Los Angeles</td>
<td>CA</td>
<td>90066</td>
<td>Diane Watson</td>
</tr>
</tbody>
</table>

Table 2: Sample constituent postal addresses

In summary, decision trees are effective because they enable us to issue intelligent predictions about recurring and new hints, accomplishing the latter by learning which attributes of the hint are the most informative and ranking them accordingly. Note that, in the worst case where the input contains only one attribute or when each hint maps to a unique value, decision-tree based prediction reduces to simple caching. Thus, decision trees represent a means for prediction that trades infrequent inaccuracies for space efficiency and the ability to predict from novel hints.

Prediction Using Finite State Devices

While predictions based on decision trees allow us to predict previously seen data based on new hints, they do not enable us to make novel predictions – that is, they do not enable us to predict values that we have never otherwise seen occur during execution. However, if we model the problem of prediction as one of translation, we can issue novel predictions.

Natural language processing research often relies on finite state devices to accomplish translation (Knight and Al-Onaizan 1998). One finite state device of interest is the subsequential transducer. This device is a state transition graph that consists of nodes (states) connected by arcs (transitions) labeled x/y where x is the input string and y is the output string. The special character # indicates that no output is generated, the character ? refers to any symbol other than those appearing on any remaining arcs for that node, and the # character corresponds to the end of the input. Arcs labeled with only output are also possible.

To understand why transducers are useful and how they can be applied, consider again the RepInfo plan in Figure 1. As the figure shows, three separate wrapper calls are required to obtain the funding graph we want. First, a representative name is used to locate the OpenSecrets member URL. This result allows us to locate the funding URL, the web page that contains the graph we want. Finally, we need to fetch that page in order to identify the URL of the graph image. This costly interleaving of navigation and retrieval is commonplace for information agents, particularly those that query web sources.

However, in looking at the data retrieved, we see that it is surprisingly predictable. As Table 3 shows, although the representative name does not correspond to the member URL, the member URL does correspond to the funding URL. Specifically, the latter can be generated from the former by replacing summary.asp with sector.asp (more precisely, replacing "summary" with "sector"). Thus, we could build a transducer that translates a member URL...
into a funding URL, enabling us to make novel predictions (i.e., predict funding URLs for new member URLs).

One such transducer that accomplishes this is shown in Figure 4. Following states 0 through 5, we can see how a member URL is translated into a funding URL. For example, the “http://www.opensecrets.org/” part of each member URL causes state 0 to be maintained (although state 1 is briefly visited for each “s”, since no “u” follows, control is returned to state 1). Meanwhile, each input symbol is copied to the output. Finally, the “s” of “summary” invites state 1 and the “u” is replaced by “e”, leading to state 2. Transition to and at state 3 enable “mma” to be replaced by “cto”. Transition to state 4 occurs upon “r” and the other arc at this state removes the “y” from “summary” and the cycle at this state enables the remainder of the URL to be appended.

### Figure 4: Transducer for member URL to funding URL

At this point, it is tempting to ask: is a transducer simply just a more inefficient means for saying replaced word with word? Although there is no question that transducers can result in word-level replacement, as shown in Figure 4, they permit a far more powerful way of describing – at a letter-level – how replacement occurs. For example, for a source value “Marina del Rey” that is embedded later in an encoded URL as “Marina+Del+Rey”, we can learn the transducer shown in Figure 5, which captures the higher-level notion of replace all spaces with plus signs. Thus, transducers are a more powerful way to describe how letters from a source value can be used to generate a target value.

### Figure 5: Transducer that replaces spaces with plus signs

#### A Unifying Learning Algorithm

In this section, we present an incremental learning algorithm called RETROSPECT that incorporates the value prediction methods above.

In general, RETROSPECT uses previous hint and answer tuples to incrementally learn how to construct prediction tuples. Specifically, the algorithm learns a predictor for each attribute of the answer tuple. This predictor is either a decision tree or a transformation rule, the latter possibly including a transducer. For example, based on the type of data in Table 2, RETROSPECT learns a decision tree for predicting federal official given a street address. In contrast, based on the data in Table 3, the algorithm learns a transformation rule (that includes a transducer) for predicting Funding URL given a Member URL.

RETROSPECT is applied at the point of speculation (e.g., by the Speculate operator) for each hint and answer tuple pair it receives. The algorithm uses the first pair to initialize all of its predictors as transformation rules. Each rule describes how to use tokens of the hint tuple to derive a value for a particular attribute of the answer tuple. This allows speculation to occur for the very next tuple.

### FUNCTION Retrospect

**INPUT:** old-predictor, hint-tuple, answer-value, predicted-value

**OUTPUT:** new-predictor

```java
if old-predictor is NULL
    new-predictor ← create transformation-rule
else
    new-predictor ← NULL
if predicted-value <> answer-value
    if old-predictor was a transformation-rule
        new-predictor ← refine transformation-rule hypothesis
    else
        new-predictor ← create decision-tree
        add (hint, answer-value) example to new decision-tree
    if not refined
        return new-predictor
```

Future answer tuple values are then first compared with the prediction tuple value issued. If they match (i.e., an accurate prediction), nothing is done – the initial transformation rule was correct. If not, RETROSPECT attempts repair. Repairing a transformation rule involves choosing a new hypothesis that describes the current situation as well as those prior. If no refinement exists, the method of prediction is changed to classification – thus, a decision tree is employed. From then on, all future incorrect predictions are handled via incremental decision tree refinement – for example, using a method similar to that described by (Utgoff 1989).

Transformation rules require further explanation. Each rule is simply a recipe for how to compose a particular value of the prediction tuple from the hint tuple. Rules consist of an ordered set of operations that progressively compose the answer value string. Table 4 shows all possible operations and what they concatenate.

### Table 4: Transformation operations

Building a transformation rule for a particular hint tuple and answer value pair then involves three basic steps:

1. Tokenizing the hint tuple and answer values
2. Finding the best alignment between the token sets
3. Using that alignment to build a transformation rule

The first step is tokenization. Hint tuples are naturally tokenized by their attribute boundaries. However, certain hint attributes, as well as the answer value, may require further domain-dependent tokenization. With web-based information gathering plans, we have found it necessary to support HTTP URL tokenization. For example, consider tokenization of the following hint and answer tuple pair:

### Table 3: The relationship between official name and the resulting OpenSecrets member and funding URLs
The second step of building a transformation rule is to find an alignment between the tokenized hint tuple and answer value. An alignment describes a mapping from hint to answer value tokens, using one of two kinds of edges. *Exact* edges are used to describe a complete replication of a hint token while a *close* edge describes an approximate replication. "Closeness" occurs when some hint and value are not equal, but some threshold percentage $\rho$ of characters from the hint exist in the answer token. For example, one possible alignment of the above hint/answer pair is:

```
| Close | http://foo.com/bar.cgi?c=Marina+del-Rey&s=CA |
```

In this example, "Marina del Rey" and "Marina+del-Rey" are close in that the hint token contains all of the characters in the answer value, but the two are not equal.

There may exist several possible alignments and thus several possible transformation rules. Any method of choosing one hypothesis over another can be used, since all are capable of generating the target value. However, alternative alignments are not discarded; they may be used later during refinement.

Once an alignment has been chosen, the transformation rule can be built. This simply involves iterating through each token of the answer value and adding one of the operations in Table 4 to the rule, based on the alignment (or lack of) with the tokenized hint. When a value token cannot be aligned to a hint token, *Append* is chosen. When an exact match exists, *Copy* is chosen. Finally, *Transduce* is used for close matches. Each Transduce is associated with a transducer that describes how to translate the hint token into the value token. Learning a transducer can be done via the approach suggested by (Oncina et al. 1993) or by simpler, approximation algorithms.

**Hypothesis refinement.** When a predicted value does not match the real value and the current method of prediction is transformation, RETROSPECT can potentially repair the old rule through hypothesis refinement. As described earlier, building a transformation rule requires choosing from a set of hypotheses $H = \{h_0, h_1,..., h_i\}$. During refinement, a new set $H'$, is developed for the hint/answer example that violates the existing rule and this new set is used to filter out incompatible hypotheses from the previous set via $H = H \cap H'$. If $H <\not\in$ $\emptyset$, a new hypothesis is chosen from the resulting $H$, otherwise refinement is deemed impossible.

**Example:** *Learning ReplInfo Predictors.* As another example of RETROSPECT, let us consider how it learns the latter two predictors in Figure 3. That is, we want one that is able to predict member URL given the name of the official and one that predicts the funding URL given member URL. Our discussion will assume that the examples seen during incremental learning are the same as those in Table 3 (Boxer, Feinstein, and Harman).

Consider the first predictor. For the first example, the hint "Boxer" is followed by the member URL. As RETROSPECT specifies, a transform rule is the form of the initial hint. However, since there is no alignment between hint and answer tuples (i.e., they share no common tokens) the resulting transformation rule is:

1. APPEND ("http://www.opensecrets.org/summary.asp?CID=N00006692...2002")

The next example (Feinstein) violates this rule and, since there are no alternatives that explain both, refinement is impossible. Thus, the algorithm reverts to a decision tree to learn the relationship between official name and URL.

Now consider learning the predictor of funding URL based on member URL. As with the previous predictor, RETROSPECT first builds a transformation rule. However, unlike the previous predictor, the alignment:

```
```

is possible and results in the (simplified) rule:

1. COPY (hint tokens 0-2)
2. TRANSDUCE (hint token 3, $\tau_1$)
3. COPY (hint tokens 4-11)

In this case, the $\tau_1$ transducer is similar to the one in Figure 4. Future examples (Feinstein, Boxer, etc) validate this rule and no further refinement is necessary.

Thus, after learning the predictors for the data in Table 3, the system can then use the name of a federal official to predict the member URL and subsequently use this speculative member URL to predict the funding URL. Thus, both predictions can be issued in parallel.

**Preliminary Results**

To begin evaluating the efficiency and accuracy of RETROSPECT, we have initially compared it to a prediction method based on caching for the modified ReplInfo plan in Figure 3. Specifically, we conducted three tests – one for each speculative opportunity in Figure 3 – and compared both approaches. These three tests are summarized below:

### Test | Hint | Prediction
---|---|---
| P1 | street address | federal official |
| P2 | federal official | member URL |
| P3 | member URL | funding URL |

Each test involved drawing $s_{\text{train}}$ training examples and $s_{\text{test}}$ testing samples from the same larger pool of size $s_{\text{pool}}$.

The P1 pool consisted of 5000 addresses in a distribution roughly equivalent to that of the US population (e.g., there were more addresses from New York and California than Wyoming). For P2, the pool was the set of all US senators and representatives (435 + 100 = 535). The P3 pool consisted of all 535 OpenSecrets member URLs. The predictors learned by RETROSPECT were the same as described earlier: P1 and P2 were decision trees while P3 was a transformation rule. Table 5 shows the accuracy and efficiency of RETROSPECT as compared to caching.

<table>
<thead>
<tr>
<th>Test</th>
<th>$s_{\text{pool}}$</th>
<th>$s_{\text{train}}$</th>
<th>$s_{\text{test}}$</th>
<th>Accuracy</th>
<th>Size</th>
<th>BITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>5000</td>
<td>1000</td>
<td>1000</td>
<td>20.42%</td>
<td>458208</td>
<td>85.48%</td>
</tr>
<tr>
<td>P2</td>
<td>535</td>
<td>100</td>
<td>100</td>
<td>17.07%</td>
<td>91840</td>
<td>16.07%</td>
</tr>
<tr>
<td>P3</td>
<td>535</td>
<td>100</td>
<td>100</td>
<td>16.78%</td>
<td>117600</td>
<td>99.90%</td>
</tr>
</tbody>
</table>

Table 5: Preliminary results comparing RETROSPECT to caching.
For P1 and P3, RETROSPECT was substantially more accurate and space efficient than caching. However, in P2 – where there was only one attribute by which to classify – RETROSPECT degenerated into a tree with one branch. Even under this worst-case classification scenario, however, use of RETROSPECT was roughly equivalent to caching.

Related Work

Learning to speculatively execute programs has been well-studied in computer science. Historically, computer architecture research has largely focused on branch prediction – which involves predicting control, not data. Recently, hardware-level value prediction has received some attention in the literature; both (Lipasti et al. 1996) and (Sazeides and Smith 1997) provide good overviews of current approaches. However, in general, the techniques that can be used for hardware-level value prediction tend to be limited by resource constraints.

To the best of our knowledge, this is the first paper that discusses learning value predictors for the speculative execution of information gathering plans. (Hull et al. 2000) describe use of speculative execution in a decision-flow framework, but speculation in that system is control-based and there is no learning involved. There is an interesting relationship between our work and (Shanmugasundaram et al. 2000), which describes how Niagara executes plans based on partial results to combat the performance penalties of blocking operators (like NEST). Both describe plan execution based on unconfirmed results; however, the partial results approach deals with data that has been generated by other operators in the system (no learning involved) and applies to consumers of aggregate operators, whereas the approach here describes how to predict intermediate data and can be applied anywhere within a plan (although it would most commonly occur after a network read). The partial results approach is also similar to past work in optimizing aggregate queries (Hellerstein et al. 1997) and approximate query answering. All of these efforts are related to the notion of speculative execution in that they attempt to increase performance by computing based on unconfirmed results. However, unlike speculative execution, none generate data predictions to accomplish that execution (i.e., they leverage real results, but in partial form) and thus none of them need to synthesize data.

Conclusion

Successful speculative execution of information gathering plans is fundamentally linked with the ability to make good predictions. In this paper, we have described how two simple techniques – decision trees and transducers – can be applied to the problem. Our initial experimental results have shown that such predictors are not only more space efficient than simple caching schemes but that they are also capable of issuing predictions for novel hints. We believe that a bright future exists for data value prediction at the information gathering level, primarily because of the potential speedup enabled by speculative execution and because of the availability of resources (i.e., memory) that exist at higher levels of execution, enabling more sophisticated machine learning techniques to be applied.

References


Data Management for Peer-to-Peer Computing: A Vision

Philip A. Bernstein1,2, Fausto Giunchiglia3, Anastasios Kementsietsidis4, John Mylopoulos3, Luciano Serafini5, and Ilya Zaihrayeu2

Abstract. We motivate special database problems introduced by peer-to-peer computing and propose the Local Relational Model (LRM) to solve some of them. As well, we summarize a formalization of LRM, present an architecture for a prototype implementation, and discuss open research questions.

1. Introduction

Peer-to-peer (hereafter P2P) computing consists of an open-ended network of distributed computational peers, where each peer can exchange data and services with a set of other peers, called acquaintances. Peers are fully autonomous in choosing their acquaintances. Moreover, we assume that there is no global control in the form of a global registry, global services, or global resource management, nor a global schema or data repository. Systems such as Napster and Gnutella popularized the P2P paradigm as a version of distributed computing lying between traditional distributed systems and the web. The former is rich in services but requires considerable overhead to launch and has a relatively static, controlled architecture. The latter is a dynamic, anyone-to-anyone architecture with little startup costs but limited services. By contrast, P2P offers an evolving architecture where peers come and go, choose whom they deal with, and enjoy some traditional distributed services with less startup cost.

We are interested in data management issues raised by this paradigm, where each peer may have data to share with other peers. For simplicity, we assume that each peer’s database is relational. Since the data residing in different databases may have semantic inter-dependencies, we allow peers to specify coordination formulas that explain how the data in one peer must relate to data in an acquaintance. For example, the patient database of a family doctor and that of a pharmacy may want to coordinate their information about a particular patient, the prescriptions she has received, and the dates when these prescriptions were filled. Coordination may mean something as simple as propagating all updates to the Prescription and Medication relations, assumed to exist in both databases. In addition, we’d like a query expressed with respect to one database to be able to use relevant databases at acquaintances, acquaintances of those acquaintances, and so on. To accomplish this, we expect the P2P data management system to use coordination formulas for recursively decomposing the query into sub-queries that are evaluated with respect to the databases of acquaintances. Coordination formulas may also act as soft constraints or guide the propagation of updates. In addition, peers need an acquaintance initialization protocol where two peers exchange views of their respective databases and agree on levels of coordination between them. The level of coordination should be dynamic, in the sense that acquaintances may start with little coordination, strengthen it over time with more coordination formulas, and eventually abandon it when tasks and interests change.

In such a dynamic setting, we cannot assume the existence of a global schema for all databases in a P2P network, or even those of all acquainted databases. Moreover, peers should be able to establish and evolve acquaintances, preferably with little human intervention. Thus, we need to avoid protracted tasks by skilled database designers and DBAs required by traditional distributed and multi-database systems [9,10].

This paper introduces the Local Relational Model (LRM) as a data model specifically designed for P2P applications. LRM assumes that the set of all data in a P2P network consists of local (relational) databases, each with a set of acquaintances, which define the P2P network topology. For each acquaintance link, domain relations define translation rules between data items, and coordination formulas define semantic dependencies between the two databases. The main goals of the data model are to allow for inconsistent databases and to support semantic interoperability in the absence of a global schema [13].

1 © 2002 Bernstein, Giunchiglia, Kementsietsidis, Mylopoulos, Serafini, Zaihrayeu.
2 Microsoft Corporation, One Microsoft Way, Redmond WA, 98052-6399. philbe@microsoft.com
3 University of Trento, 38050 Povo, Trento, Italy. {fausto,ilya}@dit.unitn.it.
4 University of Toronto, Toronto, Canada, M5S 3H5. {jm, tasos}@cs.toronto.edu
5 ITC-IRST, 38050 Povo, Trento, Italy. serafini@itc.it
The main objectives of this paper are to introduce the LRM through examples and to identify a set of open research questions on its design and implementation. Section 2 presents a motivating scenario. Section 3 sketches a formalization of LRM. Section 4 offers a preliminary architecture for an LRM-based system and relates it to past work, while conclusions appear in section 5.

2. A Motivating Scenario

Consider, again, the example of patient databases. Suppose that the Toronto General Hospital owns the TGHDB database with schema:

<table>
<thead>
<tr>
<th>Relation</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patient(ViewGroup,Sex,Age,PatRecord)</td>
<td>PatientInfo(ViewGroup,Record)</td>
</tr>
<tr>
<td>Treatment(ViewGroup,Desc,PhysID)</td>
<td>Medication(ViewGroup,Drug,Dose,Desc)</td>
</tr>
<tr>
<td>Admission(ViewGroup,Desc,ProblemDesc,PhysID)</td>
<td>Event(ViewGroup,Date,Desc)</td>
</tr>
</tbody>
</table>

The database identifies patients by their hospital ID and needs to track admissions, patient information obtained from external sources, and all treatments and medications administered by the hospital staff.

When a new patient is admitted, the hospital may want to establish immediately an acquaintance with her family doctor. Suppose the view exported by the family doctor DB (say, DavisDB) has schema:

<table>
<thead>
<tr>
<th>Relation</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prescription(ViewGroup,Med,Dose,Quantity,Date)</td>
<td>Visit(ViewGroup,Date,Purpose,Outcome)</td>
</tr>
</tbody>
</table>

Figuring out patient record correspondences (i.e., doing object identification) is achieved by using the patient's Ontario Health Insurance # (e.g., OHIP# = 1234). Initially, this acquaintance has exactly one coordination formula which states that if there is no patient record at the hospital for this patient, then the patient's record from DavisDB is added to TGHDB in the PatientInfo relation, which can be expressed as:

\[
\forall \text{fn} \land \forall \text{ln} \land \forall \text{sex} \land \forall \text{pn}. (\text{DavisDB} : \text{Patient}(1234, \text{fn}, \text{ln}, \text{pn}, \text{sex}) \rightarrow \\
\text{TGHDB} : \exists \text{tghid} \exists \text{n} \forall \text{a}. (\text{Patient}(\text{tghid}, 1234, \text{n}, \text{sex}, \text{a}, \text{Davis}, \text{pr}) \land \text{n} = \text{concat} (\text{fn}, \text{ln})))
\]

When TGHDB imports data from DavisDB, the existentially quantified variables tghid, n and a must be instantiated with some concrete elements of the TGHDB database. This amounts to generating a new TGH# for tghid, inserting the Skolem constant <undefined> for a (which will be further instantiated as the patient's age) and generating name n by concatenating her first name fn and last name ln contained in DavisDB. Later, if patient 1234 is treated at the hospital for some time, another coordination formula might be set up that updates the Event relation for every treatment or medication she receives:

\[
\forall \text{d} \forall \text{desc}. (\text{TGHDB} : \exists \text{tid} \exists \text{tghid} \exists \text{pid} \exists \text{sex} \exists \text{a} \exists \text{pr}. (\text{Treatment}(\text{tid}, \text{tghid}, \text{d}, \text{desc}, \text{pid}) \land \\
\text{Patient}(\text{tghid}, 1234, \text{n}, \text{sex}, \text{a}, \text{Davis}, \text{pr}) \rightarrow \text{DavisDB} : \forall \text{d}. (\text{sd} \leq \text{d} \leq \text{ed} \rightarrow \exists \text{desc}(\text{Event}(1234, \text{d}, \text{desc}) \land \text{desc} = \text{concat} (\text{drug}, \text{dose}, \text{at TGHDB})))
\]

This acquaintance is dropped once the patient's hospital treatment is over.

Along similar lines, the patient's pharmacy may want to coordinate with DavisDB. This acquaintance is initiated by DavisDB when the patient tells Dr. Davis which pharmacy she uses. Once established, the patient's name and phone are used for identification. The pharmacy database (say, AllenDB) has the schema:

<table>
<thead>
<tr>
<th>Relation</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prescription(ViewGroup,Med,Dose,Date)</td>
<td>Sales(ViewGroup,Med,Dose,Date,Amount)</td>
</tr>
</tbody>
</table>

Here, we want AllenDB to remain updated with respect to prescriptions in DavisDB:

\[
\forall \text{fn} \land \forall \text{ln} \land \forall \text{med} \land \forall \text{dose} \land \forall \text{qt}. (\text{DavisDB} : \exists \text{ohip} \exists \text{date} \exists \text{sex} \exists \text{pr}. (\text{Prescription}(\text{ohip}, \text{med}, \text{dose}, \text{qt}, \text{date}) \land \\
\text{Patient}(\text{ohip}, \text{fn}, \text{ln}, \text{pn}, \text{sex}, \text{pr}) \rightarrow \text{AllenDB} : \exists \text{cn} \exists \text{amount}. (\text{Prescription}(\text{cn}, \text{pn}, \text{med}, \text{dose}, \text{date}, \text{amount}) \land \text{cn} = \text{concat}(\text{fn}, \text{ln})))
\]

Of course, this acquaintance is dropped when the patient tells her doctor that she changed pharmacy.

Suppose the hospital has no information on its new patient with OHIP# 1234 and needs to find out if she is receiving any medication. Here, the hospital uses its acquaintance with an interest group of Toronto pharmacies, say TPhLtd. TPhLtd is a peer that has acquaintances with most Toronto pharmacists and has a coordination formula that allows it to access prescription information in those pharmacists’ databases. For example, if we assume that TPhDB consists of a single relation

<table>
<thead>
<tr>
<th>Relation</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prescription(ViewGroup,Med,Dose,Date)</td>
<td>Sales(ViewGroup,Med,Dose,Date,Amount)</td>
</tr>
</tbody>
</table>

then the coordination formula between the two databases might be:

\[
\forall \text{fn} \land \forall \text{ln} \land \forall \text{med} \land \forall \text{dose}. (\text{AllenDB} : \exists \text{ohip} \exists \text{date} \exists \text{sex} \exists \text{dr}. (\text{Prescription}(\text{ohip}, \text{med}, \text{dose}, \text{date}) \land \\
\text{Pharmacy}(\text{ohip}, \text{med}, \text{dose}, \text{date}) \rightarrow \text{TPhDB} : \exists \text{cn} \exists \text{amount}. (\text{Prescription}(\text{cn}, \text{pn}, \text{med}, \text{dose}, \text{date}) \land \text{cn} = \text{concat}(\text{fn}, \text{ln})))
\]
DavisDB: \( \exists \text{ohip} \ \exists \text{qt} \ \exists \text{date} \ \exists \text{sex} \ \exists \text{pr}. (\text{Prescription}(\text{ohip}, \text{med}, \text{dose}, \text{qt}, \text{date}) \text{ and } \text{Patient}(\text{ohip}, \text{fn}, \text{ln}, \text{pn}, \text{sex}, \text{pr})) \)
\( \rightarrow \) TPh: \( \exists \text{name} \ \exists \text{rep}. (\text{Prescription}(\text{name}, \text{pn}, \text{med}, \text{dose}, \text{rep}) \text{ and } \text{name} = \text{concat}(\text{fn}, \text{ln}) ) \)

Analogous formulas exist for every other pharmacy acquaintance of TPhLtd. Apart from serving as information brokers, interest groups also support mechanisms for generating coordination formulas from parameterized ones, given exported schema information for each pharmacy database.

On the basis of this formula, a query such as "All prescriptions for patient with name N and phone# P," evaluated with respect to TPhLtdDB, will be translated into queries that are evaluated with respect to databases such as AllenDB. The acquaintance between the hospital and TPhLtd is more persistent than those mentioned earlier. However, this one too may evolve over time, depending on what pharmacy information becomes available to TPhLtd.

Finally, suppose the patient in question takes a trip to Trento and suffers a skiing accident. Now the Trento Hospital database (TrentoHDB) needs information about the patient from DavisDB. This is a transient acquaintance that only involves making the patient's record available to TrentoHDB, and updating the Event relation in DavisDB.

3. A Formal Semantics for LRM

Traditionally, federated and multi-database systems have been treated as extensions of conventional databases. Unfortunately, formalizations of the relational model (such as [12]) don't apply to these extensions where there are multiple overlapping databases, which may be inconsistent and may use different vocabularies. We launch the seek for implementation solutions that address the scenario described in the previous section with a formalization of LRM.

The model-theoretic semantics for LRM is defined in terms of relational spaces each of which models the state of the databases in a P2P system. These are mathematical structures generalizing the model-theoretic semantics for the Relational Model, as defined by Reiter in [12]. Coordination between databases in a relational space is expressed in terms of coordination formulas that describe dependencies between a set of databases. These formulas generalize many forms of inter-schema constraints defined in the literature, such as [1,2,5,8,15,17].

3.1 Relational spaces

A relational space is a finite set of relational databases. Database \( i \) is associated with a logical language \( L_i \), which formalizes its schema. Abstractly, \( L_i \) is a first order language with a set of relational symbols corresponding to the relations of database \( i \), no functions symbols, and a non-empty set of constants \( dom_i \) corresponding to the domain of database \( i \). For instance, the language of DavisDB contains relational symbols such as \( \text{Patient}(x,y,z,w,v,t) \) and \( \text{Visit}(x,y,z,w) \), also the constant symbol 1234.

The content of database \( i \) is defined by a set of first order interpretations \( db_i \) of the language \( L_i \) on the domain \( dom_i \). Each interpretation \( m \in db_i \) interprets the constant symbol \( d \in L_i \) as itself and the relational symbol \( R(x_1, \ldots, x_n) \) as a finite set of n-tuples of elements of \( dom_i \), which are the tuples in the relation \( R \). To emphasize that in LRM there is no global model, we call each \( db_i \) a local database.

In LRM, there is no notion of global consistency for a set of local databases. However, we do retain a notion of local consistency. Each local database can be in a (locally) consistent or inconsistent state, and consistent and inconsistent databases can coexist in a single relational space. For instance the local databases \( db_i = \{m_i\} \), \( db_j = \{m_j,m'_j\} \), and \( db_k = \emptyset \) are respectively complete, incomplete, and inconsistent. Generally, \( db_i \) is complete if \( |db_i| = 1 \), incomplete if \( |db_i| > 1 \) and inconsistent if \( db_i = \emptyset \).

In a relational space, overlapping databases represent information about a common part of the world. This overlap has nothing to do with the fact that the same constant appears in both databases. For instance, the fact that the constant Apple appears in a database describing computers and another describing Italian agricultural products does not imply that these databases overlap. Rather, overlap is determined by the meaning of constants, i.e., when the entities denoted by constants in different databases are the same.

To represent the overlap of two local databases, one may use a global schema, with suitable mappings to/from each local database schema. As argued earlier, this is not feasible in a P2P setting. Instead, we adopt a localized solution to the overlap problem, defined in terms of pair-wise mappings from the elements of the domain of database \( i \) to elements of the domain of database \( j \). Specifically, the overlap of databases \( i \) and \( j \) is represented by two relations, called domain relations, \( r_{ij} \subseteq dom_i \times dom_j \) and \( r_{ji} \subseteq dom_j \times dom_i \). The domain relation \( r_{ij} \) represents the ability of database \( j \) to import (and represent in its domain) the elements of the domain of database
i. In many cases, domain relations are not symmetric, for instance when \( r_{ij} \) represents a currency exchange, a rounding function, or a sampling function. In a P2P setting, domain relations need only be defined for acquainted pairs of peers.

**Definition** (Relational space). A *relational space* is a pair \( <db, r> \), where \( db \) is a set of local databases on \( I \) and \( r \) is a function that associates to each \( i, j \in I \), a domain relation from \( r_{ij} \) from \( i \) to \( j \).

### 3.2 Coordination in relational spaces

Semantic inter-dependencies between local databases are expressed in a declarative language, independent of the languages supported by local databases. The formulas of this language describe properties of schemas as well as the contents of local databases in a relational space. This language is a generalization of interpretation constraints defined in [3].

**Definition** (Coordination formula). The set of *coordination formulas* \( RF \) on the family of relational languages \( \{L_i\}_{i \in I} \) is defined as follows:

\[
RF ::= i : \phi \mid RF \rightarrow RF \mid RF \land RF \mid RF \lor RF \mid \exists i : x.RF \mid \forall i : x.RF
\]

where \( i \in I \) and \( \phi \) is a formula of \( L_i \).

The basic building blocks of coordination formulas are expressions of the form \( i : \phi \), which means \"\( \phi \) is true in database \( i \\". Connectives have the usual meaning, while quantifiers require further consideration. The formula \( \forall i : x.A(x) \) should be read as "for all elements of the domain \( dom_i \), \( A \) is true". Likewise, \( \exists i : x.A(x) \), is read as "there is an element in the domain \( dom_i \) such that \( A \) is true". Notice that a variable \( x \) in the scope of a quantifier \( \exists i : x. \forall i : x \), can occur in a formula \( j : \phi(x) \), allowing quantification across domains. Specifically, we allow that within the scope of a \( dom_i \) formula, one can quantify over another domain \( dom_j \) exploiting the domain relations \( r_{ij} \) and \( r_{ji} \).

**Example.** The coordination formula \( \forall i : x. (i : P(x) \rightarrow j : Q(x)) \) is satisfied if whenever \( P(d) \) is true in database \( i \) and \( <d, d'> \in r_{ij} \), then \( Q(d') \) is true in database \( j \). Analogously the formula \( \exists i : x. (j : P(x)) \) is true if there is an element \( d \) in \( dom_i \) such that \( <d, d'> \in r_{ij} \) and \( P(d') \) is true in database \( j \). A complete formalization of truth for coordination formulas is described in [13].

Coordination formulas can be used in two different ways. First, they can be used to define constraints that must be satisfied by a relational space. For instance, the formula \( \forall l : x. (l : P(x) \lor 2 : q(x)) \) states that any object in database 1 either is in table \( p \) or its corresponding object in database 2 is in table \( q \). This is a useful constraint when we want to declare that certain data are available in a set of databases, without declaring exactly where. As far as we know, other proposals in the literature for expressing inter-database constraints can be uniformly represented in terms of coordination formulas.

Coordination formulas can also be used to express queries. In this case, a coordination formula is interpreted as a deductive rule that derives new information based on information already present in other databases. For instance, a coordination formula \( \forall l : x. (l : \exists y.p(x, y) \rightarrow 2 : q(x)) \) allows us to derive \( q(b) \) in database 2, if \( p(a, c) \) holds in database 1 for some \( c \), and \( <a, b> \in r_{12} \).

Let \( q \) represent a query posed by a user to database \( i \), and \( A(x) \) be the coordination formula body of the query. We have the following.

**Definition** (i-query). An *i-query* on a family of relational languages \( \{L_i\}_{i \in I} \), is a coordination formula of the form \( A(x) \rightarrow i : q(x) \), where \( A(x) \) is a coordination formula, \( q \) is a new \( n \)-ary predicate symbol of \( L_i \) and \( x \) contains \( n \) variables.

**Definition** (Global answer to an i-query). Let \( <db, r> \) be a relational space on \( \{L_i\}_{i \in I} \). The *global answer* of an i-query of the form \( A(x) \rightarrow i : q(x) \) in \( <db, r> \) is the set:

\[
\{d \in dom_i \mid <d, r> \models \exists i : x. (A(x) \land i : x = d)\}
\]

An intuitive reading of the above formulas is as follows. The global answer to an i-query is computed by locally evaluating in \( db \), all atomic coordination formulas \( j : \phi \) in \( A \), and by recursively composing and mapping (via the domain relations) these results according to the connectives and quantifiers that comprise the coordination formula \( A \). For instance, to evaluate the query

\[
i : P(x) \lor j : Q(x) \land k : R(x,y) \rightarrow h : q(x,y)
\]
we separately evaluate \( P(x) \), \( Q(x) \) and \( R(x,y) \) in databases \( i, j \) and \( k \), respectively. We map these results via \( r_{ih} \), \( r_{jh} \) and \( r_{kh} \) respectively obtaining three sets \( A_P \), \( A_Q \) and \( A_R \) in the domain \( \text{dom}_h \). We then compose \( A_P \), \( A_Q \) and \( A_R \) using query connectives, obtaining \( A_P \times A_Q \cap A_R \). This is the global answer to \( q(x,y) \).

4. A Preliminary Architecture for LRM

Databases in a P2P system resemble heterogeneous distributed databases, often called *multi-database systems*, e.g., Multibase [14], TSIMMIS [4], Garlic [1], and Information Manifold [8]. In most systems of this sort, a user issues queries to a global schema, and the system (called a mediator in [16]) maps the queries to subqueries on the underlying data sources. Each data source has a wrapper layer that maps subqueries into its native query language. A database designer is responsible for creating the global schema and the mappings that define its relationship to the data sources, and for maintaining the schema and mappings as data sources enter and leave the system and as their schemas evolve. At this level of detail, the overall architecture has not changed since the earliest multi-database prototypes, over 20 years ago.

Like most multi-database systems, we assume that all peer nodes have identical architectures consisting of an LRM layer running on a local data server (e.g., a DBMS). As shown in Figure 1, the LRM Layer has four modules: *User Interface* (UI), *Query Manager* (QM), *Update Manager* (UM) and *Wrapper*. UI allows a user to define queries, receive results and messages from other nodes, and control other modules of the P2P Layer. QM and UM are responsible for query and update propagation. They manage domain relations, coordination formulas, coordination rules, acquaintances, and interest groups. Wrapper provides a translation layer between QM and UM, and LIS.

Peers communicate through QM and UM using XML messages. Inter-module communication is also XML-based, shown as white arrows. The shaded arrow that connects Wrapper and LIS is different because the communication language is LIS-dependent (SQL, HTTP, …).

Strategies for query and update propagation are encoded in a set of *coordination rules*, which in most cases are expressed as ECA (Event Condition Action) rules. Coordination rules describe when, how and where a query or update must be propagated. A single formula may result in several rules. Some of these express parts of LIS as views of acquaintances, while others describe update propagations. For instance, consider the formula \( \forall x. (1 : T(x) \rightarrow 2 : S(x)) \). A reasonable coordination rule for query propagation from peer 2 might be: \( E = \text{“receive a query } Q\text{”} \), \( C = \text{“} S(x) \text{ occurs in the query } Q\text{”} \), \( A = \text{“submit the query } T(x) \text{ to peer 1”} \).

Although most of this architecture is merely a modernized version of conventional multi-databases, the P2P layer also needs to address new problems, each of which is an opportunity for future research:

- The P2P layer needs a protocol for establishing an acquaintance dynamically. This protocol can use a distributed system protocol for discovering a peer by name and establishing a session, after which each peer sends the schemas it chooses to export and with what privileges.
• After an acquaintance is established, formulas and rules are needed. The P2P layer could offer semi-automated support for generating coordination formulas, e.g., by using schema matching [11]. It might also automatically derive domain relations using data mining and other techniques; e.g., if rows of two relations have the same key, then values in matching non-key columns have the same meaning. Scrubbing rules for dirty data also need attention — a multi-database problem that's harder to cope with in a dynamic P2P setting.

• The P2P layer can use classical approaches to query processing, since coordination formulas are effectively views. However, it needs to incorporate a domain mapping logic, such as that offered by LRM. It also needs a policy on how far to propagate subqueries transitively through chains of P2P connections, which can be arbitrarily long and cyclic. In addition, standard query optimization approaches may need to be modified, e.g. with new protocols to exchange cost and utility information.

• For more effective inter-node coordination, nodes should be able to advertise their data content by giving a name and description (keywords or schema), presumably using a directory service. The P2P layer interprets this information to help users at a node create acquaintances and form Interest Groups with other nodes that have similar content.

• The problem of selecting materialized views and placing them at particular nodes becomes more difficult in a P2P scenario [7].

5. Conclusions

We have highlighted two main requirements introduced by P2P databases that distinguish them from other kinds of distributed databases. First, the mappings between databases are exclusively local, with no global schema. In support of this, we have proposed a data model for expressing such mappings between peers. Second, the set of peers is highly dynamic, requiring semi-automated solutions to problems that were formerly considered design-time, such as establishing configurations and mappings. These requirements lead to a variety of interesting, hard research problems that stretch today’s multi-database solutions beyond their current limits.

References

Abstract

Consider XML content-based document routing: a stream of XML documents are routed through a network, and routing decisions are taken based on the result of evaluating XPath predicates on these documents. Parsing XML documents and interpreting XPath expressions is the main bottleneck in such systems. We propose a novel solution to speedup the evaluation of XPath predicates based on precomputing views for the XML documents. There are both similarities and differences from the "view selection problem" in relational databases. We describe an architecture for using these views, discuss several design choices and make a brief theoretical analysis for one special case. Finally, we report some initial experiments, showing the potential for query speedup by using stream views.

1 Introduction

Consider XML content-based document routing: a stream of XML documents are routed through a network, and routing decisions are taken based on the result of evaluating XPath predicates on these documents. Parsing XML documents and interpreting XPath expressions is the main bottleneck in such systems. We propose a novel solution to speedup the evaluation of XPath predicates based on precomputing views for the XML documents. There are both similarities and differences from the "view selection problem" in relational databases. We describe an architecture for using these views, discuss several design choices and make a brief theoretical analysis for one special case. Finally, we report some initial experiments, showing the potential for query speedup by using stream views.
Distributed Queries without Distributed State

Vassilis Papadimos        David Maier
vpapad@cse.ogi.edu        maier@cse.ogi.edu

Department of Computer Science and Engineering
OGI School of Science & Engineering
Oregon Health & Science University

Abstract

Traditionally, distributed queries have been optimized centrally and executed synchronously. We outline a framework that relaxes both of these constraints using mutant query plans: XML representations of query plans that can also include verbatim XML data, references to resource locations (URLs), or abstract resource names (URNs). Servers work using local, possibly incomplete knowledge, partially evaluate as much of the query plan as they can, incorporate the partial results into a new, mutated query plan and transfer it to some other server that can continue processing. We present preliminary performance results, and discuss issues and strategies for mutant query optimization.

1 Introduction

The Internet is probably the most successful distributed computing system ever. However, our capabilities for data querying and manipulation on the Internet are primitive at best. The queries we can ask of remote servers are limited: Get the data for a given URL, or ask predefined queries using some form interface. Also, queries only involve a single client pulling data from a single server: There is no infrastructure for distributed queries. Many queries that we routinely want to ask require combining data from different data sources. We cannot always move the data pertinent to a query to a single server and do the processing there, for both technical and political reasons.

We will use a query about films, reviews, and theaters as an example. Our user, Bob, wants to see a movie tonight. Bob visits his favorite portal, BobsPortal.com, where he can ask queries about XML documents with films and showings. Bob uses some GUI front-end to come up with an XML query\(^1\) such as the following:

\[
\text{FOR } \$r \text{ in document(''film_reviews'')//review, } \$g \text{ in document(''preferences'')//genre, } \$s \text{ in document(''film_showings'')/showing[date = ''15 March 2002'']} \\
\text{WHERE } \$r/genre = \$g \text{ AND } \$r/title = \$s/title } \\
\text{RETURN } <\text{film} > \{ \$r/title \} \{ \$r/rating \} \{ \$s/theater \} </\text{film}>
\]

This query works on three XML documents: film_reviews, preferences, and film_showings. There are several kinds of magic going on here: BobsPortal.com is smart enough to know that preferences means Bob’s preferences, and film_showings only includes theaters in Bob’s town. The query also treats these documents as abstract resources; it does not mention their actual locations anywhere.

The query processor will start by translating Bob’s query into a logical query plan (Figure 1), which is a directed graph of logical query operators, such as select or join, that consume and produce sequences of tuples. A tuple contains references to XML fragments. We also have special pseudo-operators, such as document, which creates a sequence of tuples by fetching data from a URL, and display, which presents results on the client’s computer. We turn a logical query plan into a physical query plan by selecting an implementation algorithm for query operators, such as nested-loops, or hash join for logical join. A physical query plan can be executed directly by the query processor.

BobsPortal.com however, may not have the film_reviews and film_showings data locally; so how should it process the query? If it knows how to resolve these abstract documents into actual URLs, it could download them, and process the query. This approach transfers large amounts of data (every movie showing at Bob’s town, and reviews

\(^1\)The real query is more complex if we want to nest theaters inside films, and not repeat title and rating for each showing.

\(*\)Funding for this work was provided by DARPA through NAVY/SPAWAR contract N66001-99-1-8908 and by NSF ITR award IIS0086002.
for every movie currently playing anywhere), even though we only need a subset of these data. We would like to do some processing near the data sources to reduce data transfer.

In the traditional distributed query processing model, one site (the coordinator) optimizes a client’s query into a distributed query plan: a query plan where operators are annotated with the sites where they should run. The coordinator sends the sub-plans to the apprentice sites, and coordinates query execution. For query optimization to work well, the coordinator needs detailed statistics about data placement and capabilities of the other participating sites and the network. There are problems in scaling this approach to large networks of autonomous sites. A remote site may refuse to run sub-queries (it may be down, off line, or overloaded). Further, we cannot hope to maintain accurate and timely statistics on the location and characteristics of all possible Internet resources at a single centralized location.

We believe that to implement distributed queries efficiently over the Internet, we must abandon the notion of an omnipotent, omniscient coordinator able to optimize queries centrally and to oversee all aspects of their execution. We introduce a framework using mutant query plans to decentralize query optimization and execution. Mutant query plans can cope with incomplete metadata, can be optimized in a decentralized fashion, respect the autonomy and the local policies of sites they execute at, and adapt to server and network conditions even while being evaluated.

2 Mutant query plans

Mutant query plans (MQPs) [PM02] are the unit of communication in our framework for distributed XML queries. The usual way to reference data sources in an XML query is to use their URL. An MQP is a query plan graph, serialized in XML, that in addition to URLs, can refer to abstract resources using URNs, and include verbatim XML fragments. An MQP is also tagged with a network address to send its result to, once it is fully evaluated. To evaluate a mutant plan that includes URNs, we resolve these URNs to their corresponding URLs. Figure 1 is actually an MQP in disguise, with the three documents referenced as abstract resources.

Figure 2 shows how a server processes an MQP. Mutant query plans are transferred as XML documents. A server parses a MQP into a tree of query operators and constant data. Every server maintains a local catalog that maps each URN to either a URL, or to a set of servers that know more about the URN. The server resolves the URNs it knows about, then its optimizer component (re)optimizes the plan, and finds or creates sub-plans that can be evaluated locally, with their associated costs. The policy manager, at this point, makes the decision to accept or reject the mutant plan (maybe the server is overloaded, or the plan’s cost is too high). The policy manager also decides how much of a plan to evaluate locally, and passes those sub-plans to the query engine. The server then substitutes each evaluated sub-plan with its results (as an XML fragment), to get a new, mutated query plan.

If the plan is not yet fully evaluated, we must decide the next server to send it to. Again, consulting the catalog, we send the plan to a server that knows how to resolve at least one of the remaining resources. A given server does not need to know how to resolve every URN in a plan. As long as the plan eventually passes through a server that does, it can be evaluated. At some point, a server will hopefully reduce the plan to an XML document and forward it to its final destination (which may be different than its origin), or alternatively, report its failure to process the plan further.

We illustrate with Bob’s query. BobsPortal.com inserts Bob’s preferences (science fiction movies) as a constant XML fragment, to obtain the mutant plan in Figure 3(a). It does not know how to resolve film_reviews or film_showings, but does know that YourTimes.com can resolve film_showings for Bob’s town, so it sends the plan there. YourTimes.com selects today’s movies, and inserts the results as another XML fragment, as shown in Figure 3(b). The new plan is then forwarded to movies.yoohoo.com, which can resolve the film_reviews resource. Movies.yoohoo.com performs the two joins (result not shown here), and sends the final results to Bob’s computer for displaying.

Note that we can use an MQP to represent a distributed query at every stage of its processing. We can transform
the original query into a query plan and encode it in XML, obtaining an MQP; different servers can evaluate parts of the query plan and insert the intermediate results as verbatim XML fragments; and the final XML document produced as the result is again a (trivial) MQP, consisting only of XML content.

We see several kinds of services coexisting in the MQP framework. Any HTTP server is a valid MQP server (albeit of limited functionality). It only accepts point queries of the form: “Give me the data for this local URL”. A vanilla Niagara server can also function as an MQP server; one that accepts all MQPs that can be fully evaluated locally (no URNs). A full-blown MQP server handles URNs, can partially evaluate a query, and can construct and forward the mutated plan to another server for further processing.

3 Performance

We prototyped mutant query plans using the Niagara query engine [NDM99], plus three new pseudo-operators: constant encapsulates an XML fragment, resource represents a URN, and display specifies the final destination. Each server has a metadata catalog that maps each URN to a URL, or to servers that can resolve it. Our current prototype uses a greedy approach to evaluate MQPs. A server resolves as many URNs in an MQP as possible, evaluates as many operators in the MQP as it can, and then sends the mutated plan to the server that can resolve the most remaining URNs.

We also implemented a traditional, pipelined distributed plan processor for Niagara. Each plan node has a location attribute, to indicate the server the operator should run on. When a server A processing a query plan encounters a sub-plan marked for server B, it is sent to B, with a send operator on top. A replaces the sub-plan locally with a receive operator, which will connect to the corresponding send at B. The execution of the two

![Figure 2: Mutant query processing.](image)

![Figure 3: Bob’s mutant query plan: (a) at BobsPortal.com, (b) at YourTimes.com.](image)
query plans proceeds concurrently. Server B serializes tuples at the send node into XML, and transmits them to the receive node at server A, which parses them back into in-memory tuples.

We compare the two prototypes on a simple query, using the XML-encoded plays of Shakespeare\(^2\). The query asks for all the lines of Sir John Falstaff, in any play. Our setup has three identical servers, A, B, and C, with a fourth machine as the client. B stores all the comedies, while C stores all the histories and tragedies. We timed pipelined and mutant versions of our query, using two scenarios: normal load, and artificially increased load at C.

In the pipelined plan (Figure 4(a)) the client submits the query to the coordinator, A, which unions two sub-plans, running on B and C. Sub-plans scan their local plays, select Falstaff’s lines, and stream them to A. In the MQP version (Figure 4(b)), the query plan is again the union of two selects, but contains just URNs to tragedies and comedies. The client sends the MQP to A, which routes it to B with no local evaluation. B resolves the comedies URN to a set of URLs, executes that part of the plan, appends its local lines to the mutant plan, and sends it to C. C resolves the tragedies URN, executes the rest of the plan, and sends the final results to the client.

For all versions of the query, we ran each query once to warm the caches, then averaged elapsed times for ten runs. The results, and timing sequences for the three servers are shown in Figure 5. Columns A, B, and C show elapsed time on the three servers, while Client is the time, at the client, between sending the query and receiving the results. A takes negligible time to transfer a mutant plan, or start a distributed sub-plan (\(< 0.1\)s). The mutant plan has worse latency than the pipelined one, since it only works on one server at a time. Notice that the mutant plan’s total time is less than the sum of the B and C times of the pipelined plan: 13.9s vs. 16.3s under normal load. The mutant plan transfers fewer tuples: Falstaff lines from the “tragedies” are transferred once, between C and the client, instead of going from C to A to the client. **Footprint** is the sum of elapsed times for all the servers involved. The footprint of the pipelined plan is worse than for the mutant plan (“13s under normal load, “16s under heavy load). The reason is that A must wait for both apprentice sites to finish, before it finishes.

While one shouldn’t read too much into a single test, it does illustrate the trade-off available with MQPs. While performance on individual queries is somewhat worse, overall load on servers is reduced. Traditional distributed query processing requires distributing, activating and simultaneous communication with sub-plans; it requires distributed state. Our approach, in contrast, allows evaluation of distributed queries while maintaining only local state, at any point (except for brief periods to transfer MQPs).

4 Related work

*Hybrid shipping* [FJK96] combines query shipping (“send query to data”) with data shipping (“send data to query”), thus allowing query operators to execute on both clients and servers. MQPs, being simultaneously queries and data,
provide another alternative, which we might term combined shipping. The ObjectGlobe project [BKK+01] is building a distributed query processing framework over the Internet. It depends on centralized query optimization, and centralized metadata maintenance. Sahuguet et al. [SPT00] propose incorporating new execution models such as referral, chaining, leasing, and publish-subscribe into distributed query execution. We can currently implement chaining with MQPs, and are working on other mechanisms, especially publish-subscribe. Jim and Suciu [JS01] propose intensional answers (partially evaluated queries, in the form of facts and rules) in a distributed query setting, to accommodate site independence and dynamic site discovery. MQPs provide the same flexibility for queries as well as for answers.

Bonnet and Tomasic [BT98] address the problem of temporarily unavailable data sources by partially evaluating a query using the sources currently available. These partial results provide useful user feedback, and can be used to construct a parachute query, that combines the partial results with the remaining data sources to get the complete answer. We can route MQPs around unavailable data sources. A mutant plan will head for servers that can perform some work, leaving the unavailable servers for last. If those servers are still unavailable, it can either lay dormant, or head to the client with the partial results it has gathered.

5 MQP Optimizations: Consolidation, absorption, deferment

To handle mutant query plans effectively, servers must interact with their cost estimator and query optimizer components in interesting and unconventional ways. In this section, we consider local optimization on MQPs. Suppose we receive the MQP shown in Figure 6(a), where resources A, B, and C are local, while X and Y are not. We cannot evaluate any of the joins locally, since they all depend on unavailable resources, but we can rewrite this plan into an equivalent plan where we can evaluate more operators locally. We call this process consolidation.

We define an operator to be local if all its inputs are available locally, otherwise it is remote. A query plan is consolidated if at most one local operator has a remote parent. We can use join associativity and commutativity to rewrite our plan so that A, B, and C are brought together in a consolidated plan (Figure 6(b)). We can specify the consolidation process in a top-down query optimizer such as Columbia [SMB+01] using simple rewriting rules, which are special cases of associativity and commutativity. We model the locality of an operator as a logical property. Here are the five rules we need for query plans that contain only \( \Join \) and \( \text{get} \) operators. Expressions \( L \) are local, while \( R \) are remote. For example, we can derive the plan in Figure 6(b) by applying rules 1, 2, 2, 3, 1 and 2 to the original plan.

\[
\begin{align*}
R \Join L & \rightarrow L \Join R \quad (1) \\
L_1 \Join (L_2 \Join R) & \rightarrow (L_1 \Join L_2) \Join R \quad (2) \\
R_1 \Join (L \Join R_2) & \rightarrow (L \Join R_1) \Join R_2 \quad (3) \\
L \Join (R_1 \Join R_2) & \rightarrow (L \Join R_1) \Join R_2 \quad (4)
\end{align*}
\]

Figure 5: Performance results for the Falstaff query, under normal and heavy loads.
We have proved that these rules always terminate with an equivalent consolidated expression. We can extend these rules to handle various other operators. Intersection and cross-product are special cases of join. Most unary operators are also easy to handle. Can we consolidate every query plan? Unfortunately, no. Depending on the algebra, there may be expressions that we cannot consolidate. Consider the algebra that includes selection, join, and union, two relations \( L \) and \( R \) representing books in bookstores, with schema \((id, author, title, price)\), and the expression \( (\sigma_{author='X':L}) \cup \left( L \bowtie R, id=L.id, price<R.price \right) \). Suppose only \( L \) is local. We cannot consolidate the two appearances of \( L \) using this algebra. However, consolidation would be trivial if we added a ‘tee’ operator that replicates its input into multiple outputs.

Under some circumstances, we can still apply rewriting rules to increase the useful work we can perform locally, even though the resulting plan will remain non-consolidated. We can rewrite the plan in Figure 6(c) into Figure 6(d). If \(|L_1 \bowtie L_2| < |L_1|\), this rule reduces the size of the resulting MQP. We call this process absorption. A traditional relational optimizer would usually not consider such a rewriting. For MQPs, we frequently have to materialize (in space instead of time) intermediate results of query sub-plans, and anything that can shrink these results is important. While consolidation can occur before query optimization, absorption is a parallel process to query optimization, since we need the cost estimator to decide whether an absorption rewriting is truly beneficial.

Finally, even though we may be able to evaluate an operator locally, it may not pay to do so. Suppose the join between \( B \) and \( C \bowtie A \) in Figure 6(b) is a Cartesian product. Instead of evaluating it, we should just include \( B \) as verbatim XML data, to avoid inflating the resultant MQP. We call deciding which locally evaluable operators to postpone deferment. We are investigating means to cost all alternative deferments with one run of our query optimizer.

\[
R_1 \bowtie (R_2 \bowtie R_3) \rightarrow (R_1 \bowtie R_2) \bowtie R_3
\]

References


system, and then to define a specific representation of the views and evaluate their potential speedup in XML stream processing. In our particular representation, a view $V$ is given by an XPath expression, and the result of evaluating $V$ on an XML document consists of the byte offset of the node selected by $V$ in the document, or NULL if the result does not consist of exactly one node. Given a set of views to be precomputed, called a view configuration $VC$, the results of all the views in the $VC$ on a given XML document is called a header. In principle, our approach raises problems on several dimensions:

**View Selection:**
given a set of statistics on the XML streams, and the global query workload at all servers, choose a view configuration (VC) that maximizes the system’s expected throughput. The VC is then made available to all servers in the network. Optionally, we may decide to choose different view configurations for different type of XML documents; for example choose a different VC for each DTD.

**Online v.s. Offline Configuration:**
the view selection can be done offline, in which case the VC is computed before the system starts operating. The assumption here is that a central server knows the query workload, network topology, as well as statistics on the XML documents, and can choose a VC that optimizes the global performance. In the online configuration the VC is chosen dynamically by the XML data providers, based on feedback from the network. No central server is needed, but, on the other hand, a global optimum is harder to achieve. This paper restricts the discussion to offline configuration.

**Run-time Evaluation:**
given an XML document with a set of materialized views (i.e., its header), a server needs to choose a good plan to evaluate all queries in its workload in order to maximize the probability of a hit. In the case of conjunctive queries, servers may choose to evaluate conjuncts with low selectivity first and then short-circuit the evaluation whenever a condition evaluates to false.

Using materialized views in query processing is a widely applied technique in database query processing [23, 12, 7, 20, 25, 13, 5, 17]. The problem of view selection has also received significant attention as of late [1, 9, 14, 16, 22, 24, 4, 15, 18, 19, 10]. However, there are significant differences between materialized views in databases and materialized views in stream processing. First, space comes almost for free in database applications, while space is the primary limiting factor in stream processing, because the views and the XML data stream share the same network. For example, if the views are as large as the XML documents, then the network throughput for the combined stream is reduced to half of the throughput without views, more than offsetting any benefit gained from using those views. On the other hand, while the cost of updating views is a concern in databases, it is not in stream processing.

A second difference is that views in stream processing are dynamic, while in database applications they are static. Different documents may have different views and hence different headers. When the XML document is first generated, a specific header is selected that would best benefit all servers downstream in processing that particular document, and a tag is attached to specify which header the document carries. For example, in an XML document routing application where documents belong to different domains, there may be a different header for each domain. This idea can be pushed further and have multiple documents within the same domain: when a server has a miss for that document, and needs to parse the entire document, it may decide to compute another header to better help servers downstream.

This paper makes three contributions. First, it describes an architecture for using stream views, the first of its kind. Second, we provide experimental support for the potential speedup from using stream views. Finally, it explores a few directions in the design space, and provides a theoretical study for one particular choice.

## 2 Overview

We define here the problem formally and discuss a number of techniques that define the solution space.

### 2.1 Problem Setting

*XML Stream Processing Network*

\[
Q ::= G \land \ldots \land G \\
G ::= \text{Expr} \ Oprel \ \text{Const}
\]

<table>
<thead>
<tr>
<th>2</th>
<th>12</th>
<th>7</th>
<th>20</th>
<th>25</th>
<th>1</th>
<th>5</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
<td>1</td>
<td>1</td>
<td>22</td>
<td>2</td>
<td>15</td>
<td>1</td>
</tr>
</tbody>
</table>

false
2.3 Advanced Stream Processing with Views

2.2 Basic Stream Processing with Views

Dynamic Headers:

\[ C_1, \ldots, C_m \]
\[ t \in \{1, 2, \ldots, m\} \]

Nested XML Elements:

\[ \text{/news/content/address/city} \]
\[ \text{/news/content/address/country} \]
\[ \text{/news/header/agency} \]
\[ \text{/news/header/date} \]
Multiple XML Elements: 

\[ \{ p_{ij} \mid j = 1, 2, \ldots, i \} \quad s_i \quad C \quad k \quad p_{ij} \in \mathcal{P} \]

news content address country = France

3 The View Selection Problem

3.1 A Hardness Result

\[ \mathcal{P} = \{ p_1, p_2, \ldots, p_n \} \]

\[ S = \{ 1, 2, \ldots, m \} \]

\[ i = \{ i_1, i_2, \ldots, i_m \} \]

Definition 3.1 Restricted View Selection

(a) Optimization Problem: Given the sets \( \mathcal{P}, P_1, P_2, \ldots, P_n \) as defined above and a number \( k \), compute a \( C \subseteq \mathcal{P} \) of size at least \( k \) such that \( |\bigcup_j C, P_j| \) is minimized. (b) Decision Problem: Given the sets \( \mathcal{P}, P_1, P_2, \ldots, P_n \) as defined above and numbers \( k \) and \( x \), does there exist a \( C \subseteq \mathcal{P} \) of size at least \( k \) such that \( |\bigcup_j C, P_j| \leq x \).

Theorem 3.2 The Restricted View Selection Problem is NP-complete.

Proof:

\[ k \]

\[ \begin{array}{c}
11 \\
\end{array} \quad (, , ) \quad \mathcal{P} \]

\[ P_j = \{ v_1 \mid e_j \text{ is incident on the vertex } v_1 \} \quad e_j \in \mathcal{E} \quad k \quad k(k-1) \quad 2 \quad x \quad k \]

\[ = (, , ) \quad k \]

\[ (101) \]

\[ (2) \]

3.2 A Greedy Algorithm
complex versions of the problem. The greedy algorithm works by discarding, at each stage, the XPath expression \( p_x \) that is required by minimum number of remaining servers to answer their queries.

**Algorithm 1**

Greedy Algorithm for Selecting the View Configuration

1: \[ \text{for } p_j \in P \text{ do} \]
2: \[ P_j = \{ i \mid p_j \in s_i \} \]
3: \[ \text{end for} \]
4: \[ k = |P| - k \]
5: \[ \text{for } i = 1 \rightarrow k \text{ do} \]
6: \[ P \leftarrow P - \{ p \} \]
7: \[ \text{for } p_j \in P \text{ do} \]
8: \[ P_j \leftarrow P_j - P \]
9: \[ \text{end for} \]
10: \[ \text{end for} \]

The algorithm works as follows. The set \( P \) contains, at each stage, the set of XPath expressions that are candidates for being a part of the view configuration. Line 6 is the greedy decision-making step. An XPath expression \( p_x \) is chosen that is required by minimum number of servers. After \( p_x \) is discarded, we take out the servers in \( P_x \) from the list of servers for the remaining XPath expressions. This is because once \( p_x \) is discarded, a miss is going to result at all the servers in \( P_x \) and for the later iterations, we want to consider only those servers which could still have a hit. When the algorithm terminates, the set \( P \) is the required view configuration.

**4 Experiments**

Our experiments demonstrate the following: (a) when the view configuration contains all the XPath expressions used in the system, then very high speedup can be achieved. (b) significant speedup can be achieved even if a good fraction of the XPath expressions are missing from the view configuration.

Our execution environment consists of a dual 450MHz Pentium II with 1536MB memory, running Red Hat Linux 7.1. Our compiler is gcc version 2.96.2, without any optimization options. We use the Xerxes SAX parser (available from the Apache foundation [3]) to parse XML. We run each experimental run five times and we report the average. To simulate a stream of documents, we take a document and replicate it multiple times in the same file. There are 10,000 servers in this experiment. The number of queries at servers vary from 1 to 5. Each query contains a single conjunct. The XPath expressions in the conjuncts for the queries come from a Zipf distribution. Instead of distributing the queries on 10,000 different machines, the queries for all the servers reside on the same machine. And during the experiment, we evaluate the set of queries associated with each server on every replica of the document in the input file, and record the total execution time. We use the greedy algorithm from Sec. 3.2 to compute the view configuration for varying header sizes. Only the queries associated with a given server are processed in parallel. For every document-server pair, we reset the parser to the beginning of the document. We ran this experiment on documents of varying sizes. We observed similar trends. So, we report numbers for just one document, for lack of space.

Figure 1 shows the average time to process a document at a server as a function of header size. Without the header, this time is 37.99 ms, and with the header containing offsets for all the required XPath expressions, this time is 0.372 ms, giving a speedup of more than 100. Even when the header contains offsets for just two-thirds (15 out of 22) of the XPath expressions present in the system, we still get an average processing time of 4.61 ms, a speedup of 8.3. Thus, even when a fairly good fraction of XPath expressions is not present in the view configuration, a healthy speedup can be obtained by using these views.

One might argue that, using a faster parser will negate most of the speedup achieved by using the views. However, this is not the case. If we use a parser that is 10 times faster than the Xerces parser, the maximum speedup achieved does go down by a factor of 10. However, very rarely will we have the case that the header will contain the offsets for all the XPath expressions in the system. The portion of the graph in Figure 1 that we want to concentrate on is when some of the XPath expressions are missing from the view configuration. When the view configuration
5 Conclusions

References
Cryptographically Enforced Conditional Access for XML

Gerome Miklau    Dan Suciu
University of Washington
{gerome, suciu}@cs.washington.edu

Abstract

Access control for databases is typically enforced by a trusted server responsible for permitting or denying users access to the database. This server-based protection model is increasingly becoming inconvenient for web based applications. We propose encryption techniques that allow XML documents to be distributed over the web to clients for local processing while maintaining certain access controls. In particular, we focus on conditional access controls, where a user is granted access to certain data elements conditioned on the user’s existing knowledge of another part of the data. We believe such access controls are important in practice, and that enforcing them cryptographically on remote instances allows for more flexible data dissemination and processing.

1 Introduction

An access control model is used to permit or refuse access by subjects to data objects. Subjects are users, or groups of users usually defined by name, network identification or other static affiliation. For XML, objects are documents or parts of documents defined by XPath expressions. Access control in relational database systems, and most proposed XML systems, is enforced by a server that handles all data requests and strictly controls which users can access what data. While this model is sometimes also used in Web applications, it is often too restrictive. As the following examples show, there are a number of advantages to delivering remote copies of the data to clients if access control can be maintained:

Local data processing A credit card company’s accounts database is secret, but if a vendor presents a correct account number and a correct expiration date the company will provide the available credit amount. Vendors could benefit if they downloaded the entire database locally: transactions could be authenticated faster and without network access, or vendors could integrate the data with their own data and run complex queries, e.g. for marketing purposes. No information would be leaked if vendors only accessed accounts for which they know the account number and expiration date.

Privacy An information provider that sells valuable data offers different data access rights for different prices. Customer queries are answered by the provider only if the customer has sufficient access rights. But a client’s queries may reveal to the provider privileged information they prefer not to disclose¹. Remote enforcement of access control will allow the provider to deliver an entire copy of the data so that clients can process queries locally, with privacy guaranteed.

Offline browsing A Web vendor does not make its products database available to the public, but, of course, allows users to browse and query for specific products. The vendor would benefit from making the database available publicly since users could download the data to their laptop and shop offline, e.g. during a plane trip.

Peer Data Management Systems In a peer-to-peer distributed database, peers contribute data as well storage and processing resources, and members of the network can execute queries over all contributed data [12, 17]. Proposed systems require that replicas of the data be placed outside the producers’ secure domains, yet data producers need to retain some access control over their data while. Remote enforcement of access controls makes this possible, and encourages peers to share data.

We propose an approach for publishing XML data on the Web while controlling how the data is accessed. In particular, we propose a novel and flexible language of conditional access rules used to define security policy. We explain how to encrypt XML data to enforce these access controls without a trusted server, and we discuss query processing over encrypted data.

¹Private information retrieval, or the problem of allowing a subject to query a remote database without revealing information through his/her queries, was first addressed in [5].
classes. Subjects are not identified by user name or network identifier but by their knowledge. As a special case, access may be conditioned on knowledge of a private key or password in a conventional way. But more generally, subjects qualify for access to an object by virtue of their knowledge of the data. Conditional access rules specify what data values need to be presented by the subject before granting access to other data values. Subject authorization is therefore flexible and dynamic in a way not possible with conventional access classes. The flexibility of our conditional access rules distinguishes our work from other attempts to encrypt data and manage decryption keys [2, 15].

Our cryptographic enforcement of conditional policies is based on known techniques for encrypting relation tables [10]. However, we view it as critical that remote access control be implemented for XML documents, rather than relations, since relations are rarely exchanged as such.

Trusting the encryption mechanism to enforce access controls on remote replicas is a substantial departure from server-based enforcement. Some applications surely require a higher level of trust in the security mechanism and will need to rely on the server model. On the other hand, there are many applications where releasing proprietary data to customers and partners is very beneficial. Our approach targets these applications, and creates opportunities for free data dissemination and new processing models.

The paper is organized as follows. In section 2 we review encryption primitives and present a simple table encryption scheme which is the basis of our document encryption method. Section 3 presents conditional access rules. In section 4 we show how to generate an encrypted XML instance enforcing a set of conditional access rules, and then briefly discuss querying such an instance in Section 5. Overall security is reviewed in Section 6. We address related work and conclude in Sections 7 and 8.

2 Background

We present here known techniques for encrypting a relational table to enforce certain access controls. Later we adapt table encryption to enforce conditional access to XML data. We begin with definitions of encryption primitives used below [14].

Encryption Primitives Let $\mathcal{M}$ be the message space of plain text strings and $\mathcal{C}$ the cipher text space. A symmetric encryption scheme consists for each key $k$ of an encryption function $E_k : \mathcal{M} \rightarrow \mathcal{C}$ and a corresponding decryption function $D_k : \mathcal{C} \rightarrow \mathcal{M}$ with the property that for all $m \in \mathcal{M}$, $D_k(E_k(m)) = m$. An encryption scheme is secure if it is computationally infeasible to deduce $m$ from $E_k(m)$ without knowledge of $k$. Generally, functions $D$ and $E$ are publicly known, and the security of encryption rests in the key. We also use a collision free one-way function $f : \mathcal{M} \rightarrow \mathcal{C}$ such that given $m \in \mathcal{M}$, it is easy to compute $f(m)$, but given $c \in \mathcal{C}$, it is computationally infeasible to find an $m$ such that $c = f(m)$. Several symmetric encryption algorithms and candidate one-way functions are mentioned in [16]. AES is a good choice for symmetric primitives $E$ and $D$. Functions that behave like $f$ can be constructed from these, or public-key encryption techniques can be used.

Restricting Access to Relational Tables Suppose Alice has a binary relation $T[A, B]$. She wants to publish $T$, but also wants to restrict access so that a user needs to present an $A$ value before being allowed to retrieve corresponding $B$ values. In other words, she wants to allow $T$ to be published only in the context $\sigma_{A=a}(T)$, for some constant $a$. We denote this access control rule, $r$, as $T : (A \rightarrow B)$. The solution to this problem is described in [10], expanding on a method first proposed by Needham (see [8]). Using the encryption primitives described above, the access controlled table (with respect to rule $r$) is $T^ac_r$, defined as:

$$T^ac_r = \{( (f(a), E_a(b)) | (a, b) \in T \}$$

Alice publishes $T^ac_r$ (along with $f$ and $D$) instead of $T$. The tuples of $T^ac_r$ consist only of cipher text. Any subject can use $T^ac_r$ in order to access $T$ in the intended manner. For example, consider the following datalog query over table $T$:

$$Q_1(x) : = T("abc", x)$$

This query accesses the data in the way it was intended, and it can be rewritten for $T^ac$ as the following rule in datalog extended with encryption primitives:

$$Q'_1(x) : = T^ac_r(f("abc"), v), v = D_{abc}(v)$$

The re-written query computes $f("abc")$, retrieves all corresponding $B$ values $v$ from $T^ac_r$, and then decrypts them using "abc" as the key. For a more complex example, consider $Q_2$ below and its re-writing $Q'_2$:

$$Q_2(x) : = T("abc", y), T(y, z), T(z, x)$$
$$Q'_2(x) : = T^ac_r(f("abc"), v), y = D_{abc}(v),$$
$$T^ac_r(f(y), u), z = D_y(u),$$
$$T^ac_r(f(z), w), x = D_z(w)$$

$Q_2$ still accesses the data in the way it was intended: the user first evaluates $T("abc", y)$, to get a set of values for $y$, then uses each of them as a constant to
evaluate $T(y, z)$, then uses each resulting $z$ to evaluate $T(z, x)$. The rewritten query $Q_2'$ shows that it is possible to express $Q_2$ on $T^\text{ac}$, by a judicious sequence of applications of $f$ and $D$.

On the other hand it is difficult for an adversary, Mallory, to use $T^\text{ac}$ in order to execute on $T$ queries that use different access patterns. For example the following queries are hard to compute without knowing values for $x$: 

\[
Q_3(x, y) : = T(x, y) \\
Q_4(x) : = T(x, "abc")
\]

There are two known attacks to this scheme: the dictionary attack, and the guessing attack. The first applies when Mallory can generate the entire message space of $f$ which is determined by the domain of attribute $A$. For instance, when $A$ represents credit card accounts, then Mallory can enumerate all account numbers on a powerful computer and invert $f$ by brute force. The usual protection against this attack is to increase the size of the message space: take $A$ to be the account number and the expiration date. The second attack is the guessing attack: Mallory can simply probe random account numbers and dates, and discover $x$-values with some probability. We discuss the security of encryption further in Section 6.

3 Conditional Access Rules

Next we describe a flexible language of rules defining conditional access to an XML document, hence conditional access rules (CARs). For a path expression $P$, and an XML tree instance $D$, we denote by $\text{eval}(P, D)$ the node set resulting from evaluating $P$ starting from the root. A CAR $r$ has the following form:

\[
r ::= C : (\{B\} \rightarrow \{F\})
\]

Here $C$ is a single XPath expression specifying a context, $B$ and $F$ are sets of XPath expression defining, respectively, a set of required bindings, and a set of retrieved, or free values. $C$ is an absolute XPath expression (i.e. it starts from the document root); the expressions in $B$ and $F$ are relative, starting from a context node $c \in \text{eval}(C, D)$. Users specify several rules to control access to an XML document.

The intuition behind a rule is that the subtree rooted at a context node $c \in \text{eval}(C, D)$ will be hidden from the user: access is allowed only by specifying values for $B$ (a tuple of values), in which case corresponding subtree(s) $F$ will be accessible. All XPath expressions in $B$ are required to return atomic values, i.e. they have to end in $\text{text}()$ or $\text{data}()$ or in an attribute, and we assume the set $\{B\}$ to be ordered.

Figure 1 illustrates the effect of a CAR.

![Figure 1: Illustration of the effect of a CAR. Within context node c, if bound value b is provided, then the subtree rooted at free value f is accessible, except for an inner context d that may be specified by another rule.](image)

**Semantics** Let $\bar{R}$ be a set of conditional access rules, and let $D$ be an XML document tree. We denote with $x \preceq y$ the fact that node $y$ is a descendant-or-self of node $x$. In a related manner we define the “allowed” descendants to be those descendants that are not beneath any other contained context node of another rule:

\[
\text{allowed-desc}(z) = \{ x \mid z \preceq x, \neg(\exists r \in \bar{R}, r = \hat{e}'(B' \rightarrow F'), y \in \text{eval}(\hat{e}', D), z \preceq y \preceq x) \}
\]

The semantics for CARs defines a function $\text{access}_R : \mathcal{P}(\text{Dom}) \rightarrow \mathcal{P}(\text{Dom} \cup \text{nodes}(D))$, where $\mathcal{P}(S)$ denotes the set of finite subsets of $S$, with the following meaning: if the user provides values $v_1, \ldots, v_k$, then the user is allowed to see node or value $x$ if $x \in \text{access}_R(V_1, \ldots, V_k)$. Given $x \in \text{nodes}(D)$ we denote the value of $x$ with $\text{val}(x)$, and given $V \in \text{Dom} \cup \text{nodes}(D)$ we denote $\text{Val}(V) = \{ V \mid V \in \text{nodes}(D) \}$. Then $\text{access}_R$ is defined as follows:

\[
\text{access}^0_R(V) = V \cup \text{allowed-desc}(D_{\text{root}})
\]

\[
\text{access}^{(n+1)}_R(V) = \text{access}^n_R(V) \cup \{ x \mid x \in \text{access}(z), z \in \text{eval}(c[B_1 = v_1, \ldots, B_k = v_k]/f_j, D), c : \{B_1, \ldots, B_k\} \rightarrow \{f_1, \ldots, f_j\} \in \bar{R}, v_1, \ldots, v_k \in \text{Val}(\text{access}^n_R(V)) \}
\]

\[
\text{access}_R(V) = \bigcup_{n \geq 0} \text{access}^n_R(V)
\]

The function $\text{access}^n_R(V)$ computes the nodes that the user can obtain after $n$ probes to the encrypted data: to compute some node $x$ in $\text{access}^{(n+1)}_R(V)$ the
user starts by picking some values $v_1, \ldots, v_k$ already in $\text{access}_R^n$ (recall that we blur the distinction between leaf nodes and their values), uses them to bind the expressions $B_1, \ldots, B_k$ in the context, then picks some node $z$ in the result of some $F_j$: the node $x$ is any descendant of $z$, except if it is hidden from the user because some other context $\mathcal{C}$ disallows access to it. Notice that $\text{access}_R^n$ already includes all nodes in the document that are not explicitly hidden by a context.

We now provide a number of examples, based on the data pictured in Figure 2, to illustrate the flexibility of conditional access rules:

**Example 3.1** Consider the set of CARs $\tilde{R}$ containing rules:

\[
\begin{align*}
\text{hospital/patientrecords/patient} & : 
\{\text{patientid} \rightarrow \{\text{pers/name}, \text{pers/address}\}\} \\
\text{hospital/patientrecords/patient} & : 
\{\text{personal/name} \rightarrow \{\text{med/room}, \text{med/door}\}\}
\end{align*}
\]

The first rule says that within each patient context, name and address are accessible conditioned on knowledge of the corresponding patient-id. The second rule says that within each patient context, the room and floor are accessible conditioned on knowledge of the corresponding patient name. If patient-id is a key for name, and name is a key for the medical fields, then together these rules imply that a patient-id is sufficient to access name, address, room, and floor. All data not within a patient context is accessible.

**Example 3.2** Suppose the set of CARs $\tilde{R}$ contains just the rule $/ : \{\{} \rightarrow \{\}\}$. Here $C = /$, so $\text{D}_{\text{root}}$ is the only context node for this set of CARs. $B$ is the empty-set, which requires no knowledge for access to $F = \text{self}$ evaluated from the context, which is $\text{D}_{\text{root}}$. $\text{access}_R(\{\})$ contains all descendants of $\text{D}_{\text{root}}$ since there are no other contexts to consider. Thus this rule makes the entire document accessible with no conditions.

**Example 3.3** Consider the set of CARs $\tilde{R}$ containing rules:

\[
\begin{align*}
/ & : \{\} \rightarrow \{}
\text{hospital/patientrecords/patient} & : 
\{\text{patientid, medical/diagnosis} \rightarrow \{\}\}
\text{password} & : \{\} \rightarrow \{}
\end{align*}
\]

These rules make the entire XML file accessible except (1) access to patient data is conditioned on knowing a patient’s id and diagnosis, and (2) no access is given to any passwords. The last rule makes every password element a context, within which the set of free values is empty, thus protecting all passwords, regardless of other CARs in $\tilde{R}$.

**Example 3.4** Consider the set of CARs $\tilde{R}$ containing the single rule:

\[
\begin{align*}
\text{hospital} &: \langle \text{staff/physicians/physician/} \\
& \{\text{name, password, physid = $\text{PID}$} \rightarrow \\
& \{\text{patrecords/patient\[med/physid = $\text{PID}$\]\}}
\end{align*}
\]

This example shows how conventional password-authenticated access can be expressed using CARs when a password, private key, or other identifying information is included in the data. When such private information occurs in the data as bound values, it will be protected since it is within the encrypted context. The rule says that given a physician name, password, and id, the entire patient element is accessible for patients under the physician’s care. (The $\text{PID}$ notation is a shorthand requiring matching physician id’s in the bound and free parts of the rule.)

## 4 Encrypting XML

Given an XML document $D$ and a set $\tilde{R}$ of conditional access rules, we generate an encrypted document $D^e_{\tilde{R}}$ that enforces access according to $\tilde{R}$. Our strategy is to apply table encryption, as described in Section 2, to binary tables that are implied by each CAR within a context. If $r = C : (B \rightarrow F)$ is a CAR in $\tilde{R}$, and $x$ is a context node in $\text{eval}(C, D)$, then let $T_r(x)$ be the binary table which pairs the concatenation of bound data values with the concatenation of free data values. That is, $T_r(x)$ is defined by:

\[
\begin{align*}
\{b, f \mid b = b_1b_2\ldots b_k, f = f_1f_2\ldots f_i, \\
b_i \in \text{eval}(C/B_i, D), i \in [1..k], \\
f_i \in \text{eval}(C/F_i, D), j \in [1..n] \}
\end{align*}
\]

To construct $D^e_{\tilde{R}}$ we begin with $D$. Nodes that are not a context for any rule appear unchanged in $D^e_{\tilde{R}}$. But for other nodes we consider each rule $r$ for which $x$ is a context, compute $T_r(x)$ and replace $x$ and its subtree with the collection of tables $T_r(x)$ for each $r \in \tilde{R}$. The tables are represented as XML in a standard way as a list of row elements each containing two column elements (since the $T_r(x)$ are binary). These replaced elements also need to contain some metadata describing the paths to the encoded bound and free values. A similar construction can be recursively applied to nested contexts; we omit the details from this abstract.

Admittedly, this is a rather naive encryption scheme. The resulting $D^e_{\tilde{R}}$ may be large compared
with $D$, and grows with the number of conditional access rules in $R$. Furthermore, there is an opportunity for optimizing the set of CARs to produce smaller encrypted instances. Given a set of CARs $R$, we would like to find an equivalent\footnote{Two CAR sets $R_1, R_2$ are equivalent if for any XML document $D$, $\text{access}_{R_1}(D) = \text{access}_{R_2}(D)$.} set $\hat{R}$ that is minimal.

Surprisingly, we can use the theory of functional dependencies to aid minimization of a set of access rules, at least in the case of relational data. For example, given a ternary table $T[A, B, C]$ with the set of relational access rules (as in Section 2) $T : (A \rightarrow B), T : (B \rightarrow C), T : (A \rightarrow C)$ the naive encrypted version will represent consist of three binary relations, encoding $\Pi_{AB}(R), \Pi_{BC}(R), \Pi_{AC}(R)$. The first two relations, however, are sufficient. This happens because the given set of access rules is equivalent to the following: $T : (A \rightarrow B), T : (B \rightarrow C)$. For a set of relational access rules $\bar{S}$, denote $\bar{S}^{fd}$ the functional dependencies obtained from interpreting $\rightarrow$ as being a functional dependency rather than access control. For a relational schema, $T : A \rightarrow B$ has the standard meaning: $A$ functionally determines $B$ in $T$.

**Theorem 4.1** Two sets of relational access rules $\bar{S}_1, \bar{S}_2$ are equivalent iff $\bar{S}_1^{fd}$ is equivalent to $\bar{S}_2^{fd}$.

For XML, we can interpret a CAR $C : (B \rightarrow F)$ as a functional dependency according to the definition given in [3]. However, the above theorem does not hold for XML documents. We leave the interesting connection between functional dependencies and conditional access rules, along with the general CAR minimization problem, as a compelling direction for future work.

5 Query Processing of Encrypted XML

Since an encrypted, access-controlled document $D^{ac}$ may contain large portions of the original document unencrypted, it makes sense to use existing tools and technologies to query $D^{ac}$. We consider XQuery[4] a suitable general-purpose query language. For a query $q$ and an XML document $D$ we denote by $q(D)$ the answer of $q$ evaluated on $D$. The query rewriting problem is to find a new query $q'$ such that $q(D) = q'(D^{ac})$. The new query $q'$ will consist of XQuery syntax augmented with operators for the one-way function $f$ and the decryption function $D$ needed to process the data. In particular, the parts of the query that attempt to access protected contexts need to be replaced with routines that extract data from the encrypted tables represented as XML beneath the context. These routines are closely related to the rewriting examples provided in Section 2. A full treatment of the re-writing problem is omitted from this abstract.

6 Security

As we mentioned in Section 2, there are two explicit attacks on table encryption (dictionary and guessing), and these vulnerabilities are inherited by our encrypted instances. Both attacks depend on the size of the message space, determined by the product of the domain sizes of the bound attributes in a CAR. The main protection against a guessing attack is to decrease the probability of success by increasing the domain. It is worth noting that a secure server allowing similar access is also vulnerable to a guessing attack, however this can be dealt with by monitor-
ing queries and delaying the response between unsuccessful probes. In a similar manner, the computation speed of the one-way function will slow down probes to the encrypted instance but will also impede query processing.

In addition, the access controlled relational table $T^{ac}$ leaks information about $T$ in a subtle way. From $T^{ac}$ the subject can compute the number of tuples in $T$, and the number of distinct values in column $A$. The latter form of information leakage can be avoided by using an improved table encryption scheme omitted for ease of exposition.

7 Related Work

There are a number of recently proposed access control models for XML. In [6, 7, 11] XPath expressions identify fine-grained authorization objects and a server negotiates access and delivers a pruned document. A similar framework is proposed in [2] and then extended to offer remote enforcement by encrypting elements and transferring encrypted keys to users. Our work generalizes this by making access conditioned on encrypted keys, or on other knowledge of the data. The authors of [13] describe a model where provisional access policies can be defined. Such a policy grants a subject access to data provided a further action or qualification is satisfied (e.g., a subject’s activity may be logged, or a password is provided). These provisions are related to our notion of conditional access control although they work within the framework of a trusted security processor. A framework for element-level encryption of XML documents has been proposed by the W3C [9]. The IBM Security Suite[1] implements the W3C XML encryption recommendation.

8 Conclusions and Future Work

We have proposed techniques for enforcing access control without relying on a secure server, with many practical applications. Removing the server from access control enables offline query data processing, anonymous evaluation of sensitive queries, and new data sharing possibilities. In addition, remotely controlled access can still be useful in a setting where data is served to clients because server contractors may themselves be untrusted. Further, we hope our conditional access rules would make it easy for individuals to publish sensitive data in the absence of sophisticated security systems. We have identified the following key problems that require further attention: (1) minimizing a set of conditional access rules to improve the efficiency of encryption and aid users in policy definition; (2) re-writing queries over encrypted instances; and (3) quantifying the security of encrypted instances in terms of the attribute domains and properties of the encryption primitives.

Acknowledgments

Suciu was partially supported by the NSF CAREER Grant 0092955, a gift from Microsoft, and an Alfred P. Sloan Research Fellowship.

References