A Unifying Framework for Merging and Evaluating XML Information

Ho-Lam Lau and Wilfred Ng

Department of Computer Science,
The Hong Kong University of Science and Technology, Hong Kong
{lauhl, wilfred}@cs.ust.hk

Abstract. With the ever increasing connection between XML information systems over the Web, users are able to obtain integrated sources of XML information in a cooperative manner, such as developing an XML mediator schema or using eXtensible Stylesheet Language Transformation (XSLT). However, it is not trivial to evaluate the quality of such merged XML data, even when we have the knowledge of the involved XML data sources. Herein, we present a unifying framework for merging XML data and study the quality issues of merged XML information. We capture the coverage of the object sources as well as the structural diversity of XML data objects, respectively, by the two metrics of Information Completeness (IC) and Data Complexity (DC) of the merged data.

1 Introduction

Information integration, a long established field in different disciplines of Computer Science such as cooperative systems and mediators, is recognized as an important database subject in a distributed environment [5, 8]. As the networking and mobile technologies advance, the related issues of information integration become even more challenging, since merged data can be easily obtained from a wide spectrum of emerging modern data applications, such as mobile computing, peer-to-peer transmission, mediators, and data warehousing.

As XML data emerges as a de-facto standard of Web information, we find it essential to address the quality issues of integrated XML information. In this paper, we attempt to establish a natural and intuitive framework for assessing the quality of merging XML data objects in a co-operative environment. We assume that there are many XML information sources which return their own relevant XML data objects (or simply XML data trees) as a consequence of searching for a required entity from the users. To gain the maximal possible information from the sources, a user should first query the available sources and then integrate all the returned results. We do not study the techniques used in the search and integration processes of the required XML data objects as discussed in [1, 2, 3]. Instead, we study the problem of how to justify the quality of merged XML information returned from the cooperative sources.

We propose a framework to perform merging and to analyze the merged information modelled as multiple XML data objects returned from a set of XML

L. Zhou, B.C. Ooi, and X. Meng (Eds.): DASFAA 2005, LNCS 3453, pp. 81–94, 2005.

information sources. Essentially, our analysis is to convert an XML data object in an *Merged Normal Form* (MNF) and then analyze the data content of the normalized object based on a *Merged Tree Pattern* (MTP). We develop the notions of *Information Completeness* (IC) and *Data Complexity* (DC). These are the two components related to the measure of the information quality.

Intuitively, IC is defined to compute the following two features related to the completeness of those involved information sources. First, how many XML data objects (or equivalently, XML object trees) can be covered by a data source, and second, how much detail does each XML data object contain. We call the first feature of IC the *merged tree coverage* (or simply the *coverage*) and the second feature of IC the *merged tree density* (or simply the *density*).

The motivation for us to define IC is that, in reality when posing queries upon a set of XML information sources that have little overlaps in some predefined set of core labels $\mathcal{C}$, then the integrated information contains a large number of distinct XML data objects but with few subtrees or data values under the core labels, in this case the integrated information has comparatively high coverage but low density. On the other hand, if the sources have large overlaps in $\mathcal{C}$, the integrated information contains a small number of distinct objects with more subtrees or data elements under the core labels, in this case the integrated information has comparatively low coverage but high density.

The metric DC is defined to compute the following two features related to the complexity of the retrieved data items, resulting from merging data from those involved information sources. First, how diversified the merged elements or the data under a set of core labels are, and second, how specific those merged elements or data are. We call the first feature of DC the merged tree diversity (or simply the *diversity*) and the second feature of DC the merged tree specificity (or simply the *specificity*). In reality, when we merge the data under a label in $\mathcal{C}$ it may lead to a too wide and deep tree structure. For example, if most data of the same object from different sources disagree with each other, then we have to merge a diverse set of subtrees or data elements under the label. Furthermore, the merged tree structure under the label can be very deep, i.e. to give very specific information related to the label.

We assume a global view of data, which allows us to define a set of core labels of an entity that we search over the sources. As a core label may happen anywhere along a path of the tree corresponding to the entity instance, we propose a Merge Normal Form (MNF). Essentially, an XML object in MNF ensures that only the lowest core label along a path in the tree can contain interested subtrees or data elements. Assuming all XML objects are in MNF we aggregate them into a universal template called Merged Tree Pattern (MTP). We perform merging on the subtrees or data values associated with $\mathcal{C}$ from XML tree objects: if the two corresponding core paths (paths having a core label) from different objects are equal, then they can be unanimously merged in MTP. If the two paths are not equal, the conflict is resolved by changing the path to a general descendant path. Finally, if the two core paths do not exist then they are said to be incomplete, the missing node in MTP will be counted when computing IC.

The main contributions is that we establish a framework for evaluating the quality of integrated XML information. Our approach is based on a merged XML template called MTP, which is used to aggregate XML data objects from different sources. The framework is desirable for several reasons. First, the IC score is a simple but an effective metric to assess the quality of individual data source or a combination of data sources, which can serve as a basis for source selection optimization. The DC is a natural metric to assess the diversity and specificity of the subtrees under core labels. Second, MTP shares the benefits of traditional nested relations which are able to minimize redundancy of data. This allows a very flexible interface at the external level, since both flat and hierarchical data can be well presented to the users. The MTP provides for the explicit representation of the structure as well as the semantics of object instances. Finally, an XML data objects T can be converted into MNF in a linear time complexity, $O(k_1 + k_2)$, where k_1 is the number of nodes and k_2 is the number of edges in T.

Paper Organisation. Section 2 formalises the notion of integration for a set of XML data objects from a given source, which includes the discussion of the merged objects and the merged normal form (MNF). Section 3 introduces the concept of XML merged tree pattern (MTP) and illustrates how XML data objects can be merged under the MTP. Section 4 defines the components of measuring quality of integrated XML information. Finally, we give our concluding remarks in Section 5.

2 Merging Data from XML Information Sources

In this section, we assume a simple information model consisting of different XML sources, which can be viewed as a set of object trees. We introduce two notions of Merge Normal Form (MNF) and Merge Tree Pattern (MTP) in order to evaluate the merged results. Our assumptions of the information model of co-operative XML sources are described as follows.

Core Label Set. We assume a special label set over the information sources, denoted as $\mathcal{C} = \{l_1, \ldots, l_n\}$. $\mathcal{C}$ is a set of core tag labels (or simply core labels) related to the requested entity e. We term those paths starting from the entity node with tag label l_e leading to a core node with tag label the *core paths*. $\mathcal{C}$ also consists of a unique ID label, K, to identify an XML object instance of e. A user query $q = \langle e, \mathcal{C} \rangle$ is a selection of different information related to a core label, which should include the special K path. We assume heterogenicity of data objects to be resolved elsewhere, such as using data wrappers and mediators.

Key Label and Path. We assume an entity constraint: if two sources present an XML data object then we consider these objects represent the same entity in real world. The K label is important to merge information of identical objects from different sources. We do not consider the general case of FDs in order to simplify our discussion. The assumption of the K tag label is practical, since in

reality an ID label is commonly available in XML information, which is similar to the relational setting.

Source Relationships. The information source contents *overlap* to various degrees with each others, regarding the storage of XML data objects. In an extreme case, one source can be *equal* to another source, for example mirror Web sites. In the other extreme case, one source can be *disjoint* from another, i.e. no common XML data object exists in two sources, for example one source holds ACM publications and another source holds IEEE publications. Usually, *independent* sources have different degrees of overlaps, e.g. they share information of common objects. Furthermore, if all objects in one source exist in another larger one then we say the former is *contained* in the latter.

Example 1. Figure 1 shows three publication objects T_1, T_2 and T_3, all of which have a K path, key, represented in different XML object trees as shown. Each object contains different subsets of the core labels $\mathcal{C} = \{key, author, title, url, year\}$.

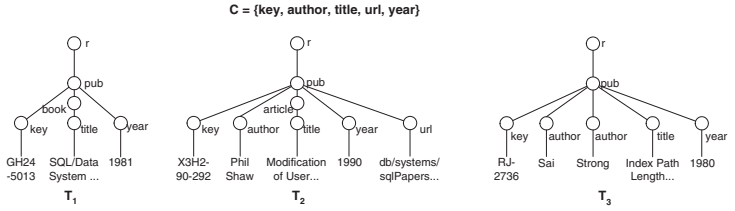

Fig. 1. A source of three XML data objects of publication records

We now consider merging the same XML data object identified by the K path. There are several scenarios arising from merging an object obtained from two different sources. (1) A core path $l \in \mathcal{C}$ of the object does not exist in either sources. (2) A core path $l \in \mathcal{C}$ of the object is provided by only one source. (3) A core path $l \in \mathcal{C}$ of the object is provided by both sources but their children under the l-node may be distinct.

The first and second cases do not impose any problems for merging, since we simply need to aggregate the existent paths in the merged result. The outcome of the merge is that there is either no information for the path l or a unique piece of information for the path l in the merged result. The last case does not bring into any problem if the children (data values or subtrees) under the core label obtained from the sources are identical. However, it poses a problem when their children disagree with each other, since conflicting information happens in the merged result. Our approach is different from the common ones which adopt either human intervention or some pre-defined resolution schemes to resolve the conflicting data. We make use of the flexibility of XML and introduce a special merge node labelled as m (m-node) as a parent node to merge the two subtrees as its children. We formalize the notion of merging in the following Definition.

Definition 1. (Merging Subtrees Under Core Labels) Let v_1 and v_2 be the roots of two subtrees, T_1 and T_2, under a core label $l \in \mathcal{C}$. Let m be the

special label for merging subtrees. We construct a subtree T_3 having the children generated by T_1 and T_2, where v_3 is a m-node whose children are defined as follows. (1) T_3 has two children of T_1 and T_2 under the root v_3, if neither v_1 nor v_2 are m-nodes. (2) T_3 has the children T_1 with T_2 being added immediately under v_3, if v_1 is under a m-node but v_2 is not. (Similar for the case if v_2 is the only m-node.) (3) T_3 has the children $child(v_1)$ and $child(v_2)$ under v_3, if both v_1 and v_2 are under m-nodes.

A *merge* operator on two given subtrees having the roots, v_1 and v_2, under a given l, denoted as $merge(v_1, v_2)$, is an operation which returns T_3 as a child under the l-node, defined according to the above conditions.

Figure 2 shows the three possible results of $merge(v_1, v_2)$, on the two subtrees, T_1 and T_2, under the core node with label $\ell \in \mathcal{C}$. The three cases correspond to the cases stated in Definition 1. We can see that the resultant subtree T_3, which has the root of a m-node, is constructed from T_1 and T_2.

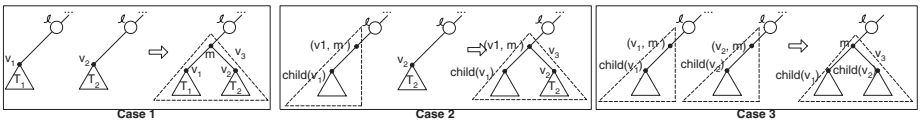

Fig. 2. The *merge* operator on two subtrees T_1 and T_2 under a core label

The *merge* operator can be naturally extended to more than two input children under a given core node with a label $l \in \mathcal{C}$. We can verify the following commutativity and associativity properties of the *merge* operator: $merge(v_1, v_2) = merge(v_2, v_1)$ and $merge(merge(v_1, v_2), v_3) = merge(v_1, merge(v_2, v_3))$. In addition, the merge operator is able to preserve the occurrences of identical data items of a core label of the same object. Our use of the merge node has the benefit that it provides the flexibility of further processing of the children under the m-node, which is independent on any pre-defined resolution schemes for conflicting data. For example, in the case of having flat data values under the m-node, we may choose an aggregate function such as *min, max, sum* or *avg* to further process the conflicting results. In the case of having tree data under the m-node, we may use a tree pattern to filter away the unwanted specific information.

One might think that it is not sufficient to define the merge operator over the same object from different sources. In fact, the merge operator has also ignored the fact that in a core path, more than one core label may occur. In order to deal with these complications, we need the concepts of Merge Normal Form (MNF) and Merge Tree Pattern (MTP) to handle general merging of XML data objects.

Definition 2. (Merge Normal Form) Let T be an XML object tree, where $\mathcal{P} \subseteq \mathcal{C}$ be the set of core labels in T and $K \in \mathcal{P}$ is the key label of T. Let us call those nodes having a core label *core nodes* and those path having a core node

core paths. A tree T is said to be in the *Merge Normal Form* (MNF), denoted as $N(T)$, if for any core paths p in T, all the ancestor nodes of the lowest core nodes of p have one and only one child.

Intuitively, the MNF allows us to estimate how much information is associated with the core labels of an entity by simply checking the lowest core label in a path. We now present an algorithm which converts a given XML object tree, T, into an MNF. By Definition 2, we are able to view $N(T) = \{p_1, \ldots, p_n\}$ as the merged normal form of T, where p_i is the core path from the root to the lowest core node in the path. Note that any particular core label may have more than one core path in $N(T)$.

MNF Generation

Input: an XML object tree T.
Output: the MNF of the tree $T_r = N(T)$

```
N(T){
1.    Let T_r = φ; P = φ;
2.    Normal(T.root);
3.    return N(T) := T_r;}
Normal(n) {
1.    for each child node h_i of n {
2.        Normal(h_i);
3.        if (label(n) ∈ C) {
4.            if (n has non-core children) {
5.                if (there exist a path, path(n') ∈ T_r, such that path(n') = path(n))
6.                    child(n') = merge(child(n'), child(n));
7.                else {
8.                    if (label(n) ∉ P)
9.                        P ∪ label(n);
10.                   T_r ∪ path(n); }}}}}
```

The underlying idea of Algorithm 2 is to visit each node in T iteratively in a depth first manner until all distinct core paths are copied as separate branches into $N(T)$. The core paths that have no non-core subtrees attached are removed. If there are two core paths ended at nodes with the same core label, we check if there exists a path, $path(n') \in N(T)$, such that $path(n') = path(n)$, their value are simply merged together, otherwise, $path(n)$ is added as a new branch. The complexity of Algorithm 2 is $O(k_1 + k_2)$, where k_1 is the number of node and k_2 is the number of edges in T. Note that the NMF of T may not be a unique $N(T)$ from Algorithm 2. However, it is easy to show that the output satisfies the requirement in Definition 2. From now on, we assume that all XML data object trees are in MNF (or else, they can be transformed to MNF by using Algorithm 2.) We now extend the merge operations on two XML data objects.

Definition 3. (Merging XML Data Object from Two Sources) Let $core(T)$ be the set of core labels in T. Given two XML object trees, $T_1 = \{p_1, \ldots, p_n\}$ and $T_2 = \{q_1, \ldots, q_m\}$, where p_i and q_j are core paths. We define $T_3 = merge(T_1, T_2)$ such that T_3 satisfies (1) $p_i \in T_3$, where $p_i \in T_1, p_i \notin T_2$. (2) $q_j \in T_3$, where $q_j \in T_2, q_j \notin T_1$. (3) $r_k \in T_3$ and $child(n_{r_k}) = merge(n_{p_i}, n_{q_j})$, where $r_k = p_i = q_j$.

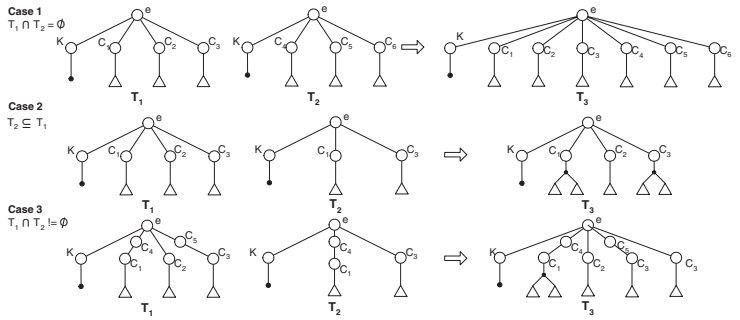

Fig. 3. Merging of MNF XML trees $N(T_1)$ and $N(T_2)$

Figure 3 shows the three possibilities of merging an XML object tree from two different sources. By Algorithm 2, we can transform an XML object tree into its MNF, which can be viewed as a set of basic core paths. We denote S as a set of XML object trees in MNF, $S = \{T_1, T_2, \ldots, T_n\}$. We further develop a template for general merging, called the *Merge Tree Pattern* (MTP), which is used to merge the information of a given set of normalized object trees obtained from different sources. Essentially, we perform merging the children of the basic core paths iteratively within MTP.

Definition 4. (Merge Tree Pattern) Let $core(T)$ denote the set of basic core labels in T. Let $\mathcal{T} = \{T_1, \ldots, T_n\}$. A Merge Tree Pattern (MTP) is a tree template obtained by combining the trees in $\mathcal{T}$. An MTP is generated according to the following algorithm. We say that two basic core paths are *mismatched*, if the two given paths both end at the same core label but they have different lists of core nodes along the basic core path. The child of each leaf of the basic core path in the MTP is a list of elements which store the data corresponding to the basic core path. We also define $desc(n)$ to be the descendant axis of the node n. For example, given $path(n) = r/a/b/c/d$, we have $desc(n) = r//d$.

Example 2. Figure 4 demonstrates the generation of MTP with three XML trees in MNF forms, T_1, T_2 and T_3. We can check that the core path for "*title*" are different in T_1, T_2 and T_3, therefore, we represent it as the descendant axis "$r//title$". The child of the core label "*author*" in T_2 is a subtree of non-core labels, in MTP, we insert a labelled pointer $(author, 1)$ to indicate it. Note that the child of the core label "*author*" subtree is a subtree having the root m-node. The list in Algorithm 4 does not store the whole tree structure, we only need to insert a labelled pointer $(m, 2)$ directed to the required subtree as shown.

MTP Generation

Let $T_i = \{path(n_{1_i}), \ldots, path(n_{p_i})\}, 1 \leq i \leq n$
and $\mathbf{T} = \{path(m_1), \ldots, path(m_q)\}$
Input: a set of trees in MNF, $\mathcal{T} = \{T_1, \ldots, T_n\}$
Output: the MTP of T
$MTPGen(T)\{$
1. **Let T** $= T_1$;
2. For each T_i in $\mathcal{T}\{$
4. For each leaf node, n, in T_i {
5. if ($path(n_i)$ and $path(m_j)$ are $mismatched$) {
6. $path(m_j) = desc(m_j)$;
7. $list(m_j).add(merge(value(m_j), value(n_i)))$;}
8. else {$\mathbf{T} \cup path(n_i)$;
9. $list(n_i).add(value(n_i))$; }}}
10. **return** $MTPGen(T) :=$ **T**; }

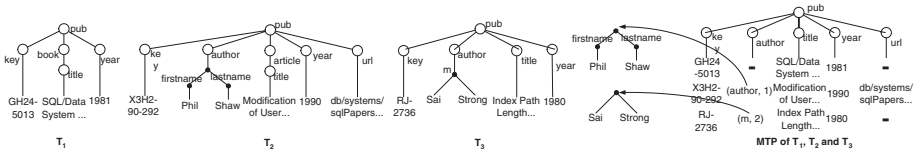

Fig. 4. Generation of MTP with three XML trees in MNF

3 Merge Operations on MTP

In order to perform merging of the entire query results from multiple sources, we define two useful merge operators on MTPs, the *join merge*, $\sqcap$, and the *union merge*, $\sqcup$.

Definition 5. (Join Merge Operator) Let P_1 and P_2 be two MTPs derived from the sources S_1 and S_2. Let $core(P_1)$, $core(P_1) \subseteq \mathcal{C}$ be the set of core labels obtained from the sources S_1 and S_2, respectively, where $K \in core(S_1)$, and $K \in core(S_2)$. Then we define the MTP $P_3 = P_1 \sqcap P_2$, such that $\exists T_1 \in S_1$ and $\exists T_2 \in S_2$ satisfies that if $r_1//K = r_2//K$, then T_3 is constructed by:
1. $path(r_3//K) := path(r_1//K)$.
2. $path(r_3//l/v_3) := path(r_1//l/v_1)$, where $l \in (core(P_1) - core(P_1))$.
3. $path(r_3//l/v_3) := path(r_2//l/v_2)$, where $l \in (core(P_1) - core(P_1))$.
4. $path(r_3//l/v_3) := desc(v_3)$ where $child(n_{v_3}) := merge(child(n_{v_1}), child(n_{v_2}))$, where $l \in (core(P_1) \cap core(P_2))$.

The intuition behind the above definition is that in the first condition we adopt the key path as the only criterion to join the tree objects identified by K obtained from P_1 and P_2. The second and third conditions state that we choose

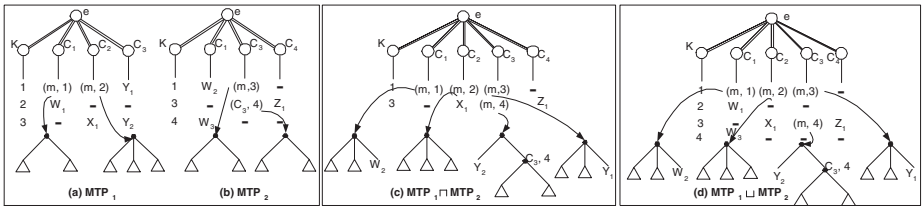

Fig. 5. The Join Merge and Join Union Operators

all the paths from the two MTPs as long as they do not overlap. The fourth condition is to resolve the conflict of having the common path in both source MTPs by using a descendant path and merging the node information.

Example 3. Figure 5(c) and 5(d) illustrates the use of the join-merge and join-union operations on P_1 and P_2.

Definition 6. (Union Merge Operator) Let P_1 and P_2 be two MTPs derived from the sources S_1 and S_2. Let $core(P_1)$, $core(P_1) \subseteq \mathcal{C}$ be the set of core labels obtained from the sources S_1 and S_2, respectively, where $K \in core(S_1)$, and $K \in core(S_2)$. Then we define the MTP $P_3 = P_1 \sqcup P_2$, such that $\exists T_1 \in S_1$ and $\exists T_2 \in S_2$ satisfies that, if $path(r_1//K) = path(r_2//K)$, then T_3 is constructed by: (1) $path(r_3//K) := path(r_1//K)$. (2)$path(r_3//l/v_3) := path(r_1//l/v_1)$, where $l \in (core(P_1) - core(P_1))$. (3) $path(r_3//l/v_3) := path(r_2//l/v_2)$, where $l \in (core(P_1) - core(P_1))$. (4) $path(r_3//l/v_3) := desc(v_3)$ where $child(n_{v_3}) = merge(child(n_{v_1}), child(n_{v_2}))$, where $l \in (core(P_1) \cap core(P_2))$. Or else T_3 is constructed by $path(r_3//l/v_3) := path(r_1//l/v_1)$ or $path(r_3//l/v_3) := path(r_2//l/v_2)$.

Notably, the union merge operator can be viewed as a generalized from of the full-outer join in relational databases [8].

4 Quality Metrics of Merged XML Trees

We describe two measures of *information completeness* and *data complexity* to evaluate the quality of the results of the join and union merge operators.

The *merged tree coverage* (or simply the *coverage*) of an XML source relates to the number of objects that the source can potentially return. Intuitively, the notion of coverage captures the percentage of real world information covered in a search. The problem lies in the fact that XML sources mutually overlap a different extent. We need to devise an effective way to evaluate the size of coverage.

Definition 7. (Merged Tree Coverage)

Let the MTP of the source S of a set of XML data objects be P_S and n be the total number of objects related to the requested entity e specified in a query

$q = (e, \mathcal{C})$, where $\mathcal{C}$ is the set of core labels associated with e. We define the *merged tree coverage* (or the *coverage*) of P_S with respect to q as $cov(S) = \frac{|P_S|}{n}$, where $\mid P_S \mid$ is the number of XML data objects distinguished by the object key $K \in \mathcal{C}$ stored in P_S.

The coverage score of simple objects is between 0 and 1 and can be regarded as the probability that any given real world object is represented by some objects in the source. We adopt the union merge operator proposed in Definition 6 to generate the MTP for the merged objects and determine the coverage score [4].

Example 4. Assume that there are about two million electronic computer science publications over the Web (i.e. $n = 2{,}000{,}000$). About 490,000 of these are listed in the Digital Bibliography & Library Project (DBLP) and the information is available in XML format. Table 1 shows the number of electronic publications available on the Web. The coverage scores are obtained by dividing the number of publications by 2,000,000.

Table 1. The coverage score of five electronic publication sources

Electronic Publication Source	Number of Publication	Coverage Scores
CiteSeer	659,481	0.3297
The Collection of Computer Science Bibliographies	1,463,418	0.7317
DBLP	490,000	0.2450
CompuScience	412,306	0.2061
Computing Research Repository (CoRR)	75,000	0.0375

The coverage measure for the MTP from many sources can be computed in a similar way, based on the coverage scores of individual sources. In reality, we may download the source to assess the coverage or the coverage can be estimated by a domain expert. To respond to a user query, a query is sent to multiple XML information sources. The results returned by these sources are sets of relevant XML data objects. Some data objects may be returned by more than one source. We assume that there are only three different cases of overlapping data sources.

1. The two sources are *disjoint*, which means that, according to the K label, there are no common XML data objects in the two sources. Then $cov(S_i \sqcup S_j)$ is equal to $cov(S_i) + cov(S_j)$ and $cov(S_i \sqcap S_j)$ is equal to 0.

2. The two sources are *overlapping*, meaning that, according to the K label, there are some common XML data objects in the two sources. The two sources are assumed to be independent. Then $cov(S_i \sqcup S_j)$ is equal to $cov(S_i) + cov(S_j) - cov(S_i) \cdot cov(S_j)$ and $cov(S_i \sqcap S_j)$ is equal to $cov(S_i) \cdot cov(S_j)$ (if S_i is contained in S_j).

3. One source is *contained* in another, which means that, according to the K label, all the XML data objects in one source are contained in another source. Then $cov(S_i \sqcup S_j)$ is equal to $cov(S_j)$ and $cov(S_i \sqcap S_j)$ is equal to $cov(S_i)$ (if S_i is contained in S_j).

Now, we consider the general case of integrating results returned from multiple data sources. We emphasize that the extension of the two merge operations in Definitions 5 and 6 from two sources to many sources is non-trivial,

since mixed kinds of overlapping may occur between different sources. We let $M = \bigsqcup(S_1, \ldots, S_n)$ be the result obtained from union-merged a set of sources $W = \{S_1, \ldots, S_n\}$. Let $S \notin W$. We define the disjoint sets of sources $D \subseteq W$ to be the maximal subset of W, such that all the sources in D are disjoint with S, the contained sets of sources $T \subseteq W$ to be the maximal subset of W, such that all the sources in T are subsets of S, and the independent sets of sources $I \subseteq W$ to be the remaining overlapping cases, i.e. $I = W - T - S$.

Theorem 1. The following statements regarding W and S are true.
1. $cov(M \sqcup S) = cov(M) + cov(S) - cov(M \sqcap S)$.
2. If $\nexists S_i \in W$ such that $S \subseteq S_i$, then $cov(M \sqcap S) = cov(M) + cov(S) - cov(M \sqcap S)$, or else $cov(M \sqcap S) = cov(S)$.

The *merged tree density* (or simply the *density*) of an XML source relates to the ratio of core label information provided by the source. As XML objects have flexible structures, the returned object trees from a source do not necessarily have information for all the core labels. Furthermore, a basic core node may have a simple data value (i.e. a leaf value) or a subtree as its child. We now define the density of a core label in an MTP, P_S.

Definition 8. (Core Label Density) We define the *merge tree density* (or simply *density*) of a core path p_l for some $l \in \mathcal{C}$ of P_S, denoted as $den(S, l)$, by $den(S, l) = \frac{|\{T \in P_S \mid r_T//p_l/v \text{ exists in } T\}|}{|P_S|}$, where "$r_T//p_l/v$" is a basic core path and v is the corresponding core node in T. The density of the MTP P_S, denoted as $den(S)$, is the average density over all core labels and is given by $den(S) = \frac{\Sigma_{l \in \mathcal{C}} den(S,l)}{|\mathcal{C}|}$.

In particular, a core label that has a child (a leaf value of a subtree) for every data tree of the source S has a density of 1 in its P_S. The density of a core label l that is simply not provided by any object data tree has density $den(S, l) = 0$. Core labels for which a source can provide some values have a density score in between 0 and 1. By assumption $den(S, K)$ is always 1.

Table 2. The DBLP XML table from MTP(DBLP)

key	title	author	journal	volume	year	url
tr/ibm/GH24...	SQL/Data System ...	-	IBM Publication	GH24-5013	1981	-
tr/ibm/RJ...	Index Path Length...	Sai, Strong	IBM Research Report	RJ2736	1980	-
tr/sql/X3H2...	Modification of User...	Phil Shaw	ANSI X3H2	X3H2-90-292	1990	db/systems/...
tr/dec/SRC...	The 1995 SQL Reunion...	-	Digital System...	SRC1997-018	1997	-
tr/gte/TR-026..	An Evaluation of Object...	Frank Manola	GTE Laboratories...	TR-0263-08-94-165	1994	db/labs/gte/...

Example 5. Let $\mathcal{C} = (key, title, author, journal, vloumn, year, url)$ be a simplified set of core labels of an article object. Consider the DBLP table (ignoring all core paths) extracted from MTP(DBLP) as shown in Table 2. The data in the table is the information returned from the $DBLP$ source for searching articles. The density of the core labels *title* and *url* are $den(S, title) = 1$ and $den(S, url) = 0.4$, respectively.

Similar to finding real coverage scores, density scores can be assessed in many ways in practice. Information sources may give the scores for an assessment. We may also use a sampling technique to estimate the density. For large data sources, the sampling process can be continuous and then the score can be incrementally updated to a more accurate value.

Now, we consider the general case of n data sources. We use the same set of notations M, W, T, S and I as already introduced in Section 4.

Theorem 2. The following statements regarding W and S are true.
1. $(den(M \sqcup S) = den(M,l) \cdot cov(M) + den(S,l) \cdot cov(S) - den(T,l) \cdot cov(T) - (den(S,l) + den(I,l) - den(S,l) \cdot den(I,l)) \cdot cov(S) \cdot cov(I) + (den(I,l) + den(T,l) - den(I,l) \cdot den(T,l)) \cdot cov(I \sqcap T)) \cdot \frac{1}{cov(M \sqcup S)}$
2. $den(M \sqcap S, l) = den(M,l) + den(S,l) - den(M,l) \cdot den(S,l)$.

Example 6. Assume that $DBLP(D)$ and $CiteSeer(C)$ are independent sources. Let the density scores for the *volumn* label be 1 and 0.6 respectively. The coverage score is 0.245 and 0.3297. Thus, the density score of their merged result is given by $den(D \sqcup C) = 1 \cdot 0.245 + 0.6 \cdot 0.3297 - (1 + 0.6 - 1 \cdot 0.6) \cdot 0.245 \cdot 0.3297 \cdot \frac{1}{0.245 + 0.3298 - 0.245 \cdot 0.3297} = 0.2387$. We now add the $CompuScience(S)$ and assume it is independent of $DBLP$ and $CiteSeer$. Its density of 0.8 for the *volume* label and a coverage of 0.2061. The new density score is given by: $den(D \sqcup C \sqcup S) = 0.2387 \cdot 0.5747 + 0.8 \cdot 0.2061 - (0.2387 + 0.8 - 0.2387 \cdot 0.8) \cdot 0.5747 \cdot 0.2061 \cdot \frac{1}{0.5747 + 0.2061 - 0.5747 \cdot 0.2061} = 0.4179$.

4.1 Information Completeness and Data Complexity

The notion of *information completeness* of an information source represents the ratio of its information amount to the total information of the real world. The more complete a source is, the more information it can potentially contribute to the overall response to a user query.

Definition 9. (Information Completeness) The *Information Completeness* (IC) of a source S is defined by

$$comp(S) = \frac{\text{No. of data objects associated with each } l \in \mathcal{C} \text{ in } P_S}{\mid W \mid \cdot \mid \mathcal{C} \mid},$$

where W is the total number of data objects of a real world entity and P_S is the MTP of S.

The following corollary allows us to employ coverage and density to find out the IC score. This corollary can be trivially generalised to a set of information sources using the corresponding MTP. Intuitively, the notion of IC can be interpreted as the "rectangular area" formed by coverage (height) and density (width). The following example further helps to illustrate these ideas.

Corollary 1. Let S be an information source. Then $comp(S) = cov(S) \cdot den(S)$.

Example 7. Table 2 represents the entries DBLP XML source. The table provides only five tuples with varying density. The coverage of the source is thus given by $cov(DBLP) = \frac{5}{2{,}000{,}000}$. The densities for the labels are 1, 1, 0.6, 1, 1, 1, and 0.4, respectively, and it follows that the density of the source is $\frac{6}{7}$. Thus, the completeness of $DBLP$ is $\frac{5}{2{,}000{,}000} \cdot \frac{6}{7} = \frac{3}{1{,}400{,}000}$.

We define *specificity* and *diversity* of an XML source to represent the depth and breadth of data that the source can potentially return. As the subtree of a core label may contain subtrees of flexible structures, the returned data does not contain the same amount of data, it may contain something as simple as a single textual value, or a complex subtree having many levels.

Definition 10. (Specificity and Diversity) Let P_S be the MTP of a source S of a set of XML data objects. Let $avg(d_i)$ and $max(d_i)$ be the average and maximum depth of child subtrees under the core node labelled by $l_i \in \mathcal{C}$ and D be the maximum of $\{max(d_1), \ldots, max(d_n)\}$ where $\mid \mathcal{C} \mid = n$. *Specificity* is defined by $spec(S) = \frac{\Sigma avg(d_i)}{n \cdot D}$. Similarly, we define *diversity* by $div(S) = \frac{\Sigma avg(b_i)}{n \cdot B}$, where $avg(b_i)$ and $max(b_i)$ are the average number and the maximum of children of subtrees under the core node labelled by $l_i \in \mathcal{C}$. Similarly, B be the maximum of $\{max(b_1), \ldots, max(b_n)\}$.

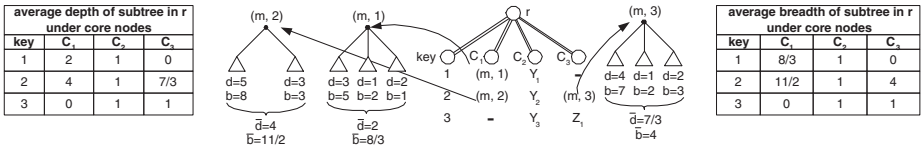

Fig. 6. Specificity and diversity of subtrees

Example 8. In Figure 6, under the core label C_1, the list contains three values: "$(m,1), (m,2), -$". The depth of core label C_1 is $d(C_1) = \frac{d((m,1))+d((m,2))+d(-)}{3} = \frac{2+4+0}{3} = 2$. Similarly, $d(C_2) = \frac{1+1+1}{3} = 1$ and $d(C_3) = \frac{0+\frac{7}{3}+1}{3} = 1.1111$. The deepest path is $(m,2)$, so we have $D = 4$. The specificity of the tree is $spec(S) = \frac{d(C_1)+d(C_2)+d(C_3)}{n \cdot D} = \frac{2+1+1.1111}{3 \cdot 4} = 0.3426$. The breadth of core label C_1, $b(C_1) = \frac{b((m,1))+b((m,2))+b(-)}{3} = \frac{\frac{8}{3}+\frac{11}{2}+0}{3} = 2.7222$. Similarly, $b(C_2) = \frac{1+1+1}{3} = 1$ and $b(C_3) = \frac{0+4+1}{3} = \frac{5}{3}$. The broadest subtree is $b((m,2))$, so we have $B = 5.5$. The diversity of the tree is $div(S) = \frac{b(C_1)+b(C_2)+b(C_3)}{n \cdot B} = \frac{2.7222+1+1.6667}{3 \cdot 5.5} = 0.3266$.

The notion of *data complexity* of an information source is employed to represent the amount of information from the source. The higher the data complexity of a source is, the richer and broader information it can potentially contribute to the overall response to a user query.

Definition 11. (Data Complexity) The *Data Complexity* (DC) of a source S is defined by $cpex(S) = spec(S) \cdot div(S)$.

Example 9. Consider the MTP in Figure 6, the data complexity, DC of the tree is $cpex(S) = spec(S) \cdot div(S) = 0.3426 \cdot 0.3266 = 0.1119$.

5 Concluding Remarks

We have proposed a framework which consists of two useful concepts, the first being information completeness (IC), which represents the coverage of data objects and the density of data related to a set of core labels, and the second being data complexity (DC), which represents the diversity and the specificity of data content for a set of core nodes associated with the search entity. The framework allows merging XML data objects obtained from different sources. We present an MNF as a standard format to unify the essential data in merged objects of an entity and an efficient algorithm to transform an XML data object into an MNF. We develop MTP as a unifying template to represent the merge of a set of XML objects. MTP serves as a basis to evaluate the IC and DC scores. We also investigated the properties of density and coverage via two merge operators in different sources that are disjoint, overlapping and independent. An important issue related to this work is how we obtain the values of various metrics of coverage, density, diversity and specificity. We suggest that this information can be derived from the data sources or from other authority corpora. For example, from the probability distribution on certain topics of CS if we compare those complete or almost complete sources we can then compute the coverage.

References

1. E. Bertino and E. Ferrari. XML and Data Integration. *IEEE Internet Computing.* 5(6):75-76, 2001.
2. V. Christophides, S. Cluet, and J. Simeon. On Wrapping Query Language and Efficient XML Integration. In *Proc. of SIGMOD Conference*, 2000.
3. Z. G. Ives et al. An Adaptive Query Execution System for Data Integration. In *Proc. of SIGMOD* , 1999.
4. D. Florescu, D. Koller, and A. Levy. Using probabilistic information in data integration. In *Proc. of VLDB*, 1997.
5. E. Lim, J. Srivastava, S. Prabhakar and J. Richardson. Entity Identification in Database Integration. In *Proc. of ICDE*, 1993.
6. A. Motro and I. Rakov. Estimating the quality of databases. In *Proc. of FQAS*, 1998.
7. F. Naumann, J. C. Freytag and U. Leser. Completeness of Information Sources. In *Proc. of DQCIS*, 2003 .
8. Elmasri and Navathe. Fundamentals of Database Systems. Addison Wesley, 3rd Edition, 1997.