

# QMatch - A Hybrid Match Algorithm for XML Schemas

Kajal T. Claypool      Vaishali Hegde  
Naiyana Tansalarak

Department of Computer Science, University of Massachusetts - Lowell  
{kajal|vhgde|ntansala}@cs.uml.edu

## Abstract

*Integration of multiple heterogeneous data sources continues to be a critical problem for many application domains and a challenge for researchers world-wide. With the increasing popularity of the XML model and the proliferation of XML documents on-line, automated matching of XML documents and databases has become a critical problem. In this paper, we present a hybrid schema match algorithm - QMatch - that provides a unique framework for combining existing structural and linguistic algorithms while exploiting additional information inherent in XML documents such as the order of XML elements to provide improved levels of matching between two given XML Schemas. QMatch is based on an extension of our previous work on QoM, a Quality of Match metric that measures the “goodness” of a match of two UML-based schemas. We now extend the QoM taxonomy to encapsulate the richness of information captured in XML Schemas and provide a qualitative and quantitative analysis of the information capacity of XML Schemas. QoM provides not only a means of tuning existing schema match algorithms to output at desired levels of matching but also provides an effective basis for a new schema match algorithm. In this paper we show via a set of experiments the benefits of QMatch over using individual structural and linguistic algorithms for schema matching, and provide an empirical measure of the accuracy of QMatch in terms of the true positives discovered by the algorithm.*

## 1 Introduction

The proliferation of the Internet and the large acceptance of the XML standard has led to a growing number of XML documents on the Web, making the comparison of the Web to a “database” closer to reality than ever before. And like

for any other database, users of this large database require fast and efficient querying to get to the desired nuggets of information. Today keyword searches such as those supported by Google [9] and HotBot [10] (and others) are the most popular form of querying the Web. However, while these queries do indeed narrow down the contents of the Web to the pertaining subset of information, they do not have the expressive power for the user to specify an exact query; and they often still require users to page through anywhere from 10 to 100 pages of results. An alternative technique is the use of traditional query languages and query processing techniques to query the Web using perhaps a language such as XQuery [18]. However, the Web does not have a fixed global schema that can be used by the query engine. Rather the Web provides a melting pot of information from many different domains. Even within a given domain, semantically similar documents very possibly have differing schemas. One of the key issues in the querying of these XML documents is thus that of matching, where in the schema of the query must be matched with the schema of the XML documents.

In this paper, to address this key issue, we present **QMatch** a hybrid match algorithm that provides a unique framework for analyzing and exploiting semantic and structural information inherent in XML schemas. The **QMatch** hybrid framework is based on **QoM** [17] - a Quality of Match metric that measures the “goodness” of a match between two entities based on the semantic (labels) and the structural (contents of the elements) information, including the children, the nesting level as well as other properties such as the order, inherent in XML schemas. Specifically, in this paper we propose (1) an *XML match taxon-*

omy that qualitatively categorizes the structural and semantic overlap between two given XML schemas; and (2) a weight-based match model that quantitatively evaluates the quality of match, assigning it an absolute numeric value. The **QMatch** algorithm uses this XML match taxonomy as a guide for the algorithm execution, and also utilizes the weight-based match model to calculate the QoM for every node in the XML Schema tree combining both structural and linguistic algorithms in the process. We present some preliminary results to provide an empirical measure of the accuracy of **QMatch** in terms of the true matches discovered by the algorithm.

**RoadMap:** The rest of the paper is organized as follows. In Section 2 we present the XML match taxonomy. Section 3 presents a cost model that quantifies the QoM between two XML Schema elements. In Section 4 we introduce **QMatch** - our hybrid match algorithm. We present our experimental evaluations in Section 5. Section 6 briefly reviews relevant literature and we conclude in Section 7.

## 2 XML Match Taxonomy - Defining a Qualitative Measure of a Match

We define the *quality of match* (QoM) metric as the measure of "goodness" of a given match. In this section we define the *XML Match Taxonomy* that classifies the information capacity of given XML schemas and based on that categorically determines the quality of match between two given entities. All examples in this section are based on the XML schemas given in Figures 1 and 2.

### 2.1 QoM Match Classification for XML Schemas

The information captured by each individual element in an XML Schema can be classified along four primary axes - its **label** L, its set of **properties** P, its **children** (subelements and attributes) C, and its nesting **level** H in the schema tree. The label, properties and level axes are regarded as atomic valued axes, while the children axis is considered to be a set-valued axis. A match along the atomic-valued axes can be classified as either an *exact* match or a *relaxed* match.

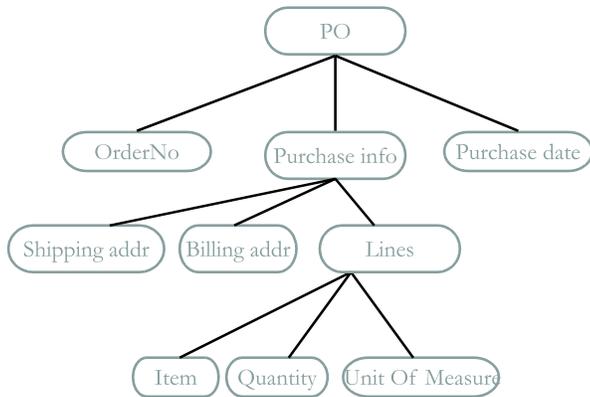
**Exact Match.** A match is said to be *exact* if the value  $v_1$  of the axis, where axis is either the label, properties, or level axis, in schema  $S_1$  is identical to the value  $v_2$  of the same axis in schema  $S_2$ . For the label axis, an exact match denotes an exact string match, a synonym match or an ontology based match obtained via a linguistic match algorithm [12, 5, 3, 4, 6]. Consider the schemas shown in Figures 1 and 2. Here the label of the element `OrderNo` in the PO schema matches *exactly* the label of element `OrderNo` in the Purchase Order schema.

For the level axis, an exact match implies that the values  $v_1$  and  $v_2$  of the axis are identical. Consider again the schemas shown in Figure 1 and 2. The elements `OrderNo` in PO and `OrderNo` in PurchaseOrder are leaf nodes existing at the same *level*. Hence these elements are said to have an exact match along the level axis.

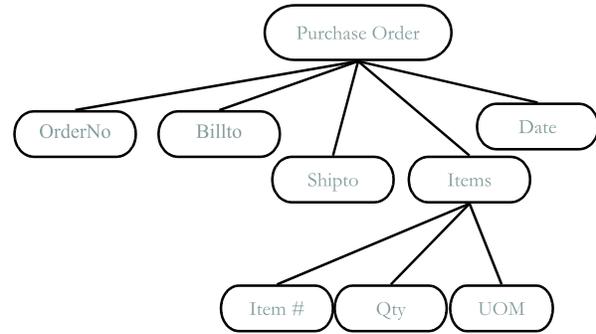
To decide the match along the properties axis we must determine the match for each individual property. An exact match for each individual property implies that the value  $v_1$  of the source property  $p$  is identical to the value  $v_2$  of the target property  $p$ . We say that a match along the properties axis is exact if there is an exact match for all constituent properties. For example, assume that the elements `OrderNo` in Figures 1 and in 2 have three defined properties, *type*, *order*, and *minOccurs*. Assuming that the values of these properties for both `OrderNo` elements are identical, that is, *type* = integer, *order* = 1, and *minOccurs* = 1 for both elements, the match along the properties axis is said to be exact.

**Relaxed Match.** A match is said to be *relaxed* if the value  $v_1$  of the axis, where axis is either the label, properties, or level axis, in schema  $S_1$  has some degree of match (but not exact) to the value  $v_2$  of the same axis in schema  $S_2$ . For the label axis a relaxed match denotes either a hypernym match or acronym match obtained via a linguistic match algorithm [12, 5, 3, 4, 6]. Consider again the schemas shown in Figures 1 and 2. Here the label of the element `Unit Of Measure` in the PO schema has an acronym match with the label of element `UOM` in the Purchase Order schema - denoting a relaxed match along the label axis.

For the level axis, a relaxed match is synonymous with no match and implies that the values  $v_1$  and  $v_2$  have differ-



**Figure 1. A Tree Representation of the PO Schema.**



**Figure 2. A Tree Representation of the Purchase Order Schema.**

ent values.

For the properties axis, a match is said to be relaxed if the consensus of the matches of the individual properties is relaxed. The match for a property is decided on an individual basis. For example a relaxed match for the *order* property implies that the order values  $v_1$  and  $v_2$  of the source and target element are not equal. On the other hand, a match between properties such as *minOccurs*, *maxOccurs*, and *type* is said to be relaxed if the property value of the source is a generalization or a specialization of the target property. For example, *minOccurs* = 0 is a generalization of the constraint *minOccurs* = 1. A full list of properties and conditions for a relaxed match appear in [8].

**Coverage Match - Total vs Partial.** The match along the last axis of information, the children axis, quantifies the level of coverage of the sub-elements and the attributes in the source element with respect to the sub-elements and the attributes of the target element. Dependent on the level of *coverage*, the match along the children axis is categorized as either *total* or *partial*. The coverage match is *total* if all children (sub-elements and attributes) of the source element have a match with some child of the target element. For example, in the schemas given in Figure 1 and 2, the element *Lines* has a *total* coverage match with the element *Items* in the target schema *PurchaseOrder* in Figure 2. A coverage match is said to be *partial* if some but not all the children of the source element have a match with children of the target element.

## 2.2 Categorizing XML Matches - The XML Match Taxonomy

Based on the QoM classifications presented in Section 2.1 we now present a taxonomy for XML matches.

**Leaf Match.** In an XML Schema, a basic element declaration provides the name of the element and a set of properties associated with it. This basic declaration holds for both simple elements and attributes, as well as constrained elements such as *restriction* elements. Leaf elements, that is elements with no children, are compared along these two primary axes of information, namely the label and properties axes. The nesting level for a leaf element is always set to 0. In keeping with the general classification, the QoM for leaf elements is categorized as either *exact* or *relaxed*. A match between two leaf elements is said to be *exact*,  $E_1 = E_2$ , if both its label and set of properties match exactly as specified in Section 2.1. For example, the match between the two leaf elements *OrderNo* in the PO schema (Figure 1) and the element *OrderNo* in the *Purchase Order* schema (Figure 2) is *exact* as their labels and properties match exactly.

A match between two leaf elements  $E_1$  and  $E_2$  is said to be *relaxed*, if either the label or any of the properties of element  $E_1$  have a relaxed match with the label and the properties of  $E_2$  respectively. The match between the leaf elements *Quantity* in the PO schema (Figure 1), and the element *Qty* in the *Purchase Order* schema (Figure 2)

is said to be *relaxed* as the label `Quantity` has a relaxed match with the label `Qty`. Their set of properties match exactly.

**Subtree Match.** We now take a look at an intermediate node, that is a non-leaf node. We consider this a *sub-tree* match as it is a comparison of two subtrees rooted at the two intermediate nodes. Sub-tree matches can be categorized based on (1) the number of children matches; (2) the quality of match of the children; and (3) the quality of match along the atomic valued axes of the root node (of the sub-tree). Thus, two non-leaf node are compared along all four axes of information – the label, properties, children and level axes.

Let us consider the children axis first. We use the classification of *total* and *partial* to compare matches along the children axis. However, quantifying the number of matching children alone does not give us an accurate assessment on the quality of match for the intermediate node along the children axis. We must also consider the quality of match (QoM) of the individual children elements themselves. Combining these together, we now define four categories of match between two given non-leaf nodes: *total exact*, *total relaxed*, *partial exact* and *partial relaxed*. A match is *total exact* if all the children of the source element have an *exact* match to some child element of the target element; *total relaxed* if all children of the source element match some child element of the target element, but one or more of these matches is *relaxed*. Similarly, *partial exact* and *partial relaxed* can be defined for the case when not all children of the source element have a match in the target element.

As a next step, we combine the QoM along the children axis with the QoM of the sub-tree root along the label, properties and level axis. The match along these atomic valued axes is classified as *exact* or *relaxed* as before. Combining these with the QoM of the children axis we now refine the *total exact*, *total relaxed*, *partial exact* and *partial relaxed* matches as follows. A match is said to be *total exact* if there is an exact match along the label, properties and level axis, and a total exact match along the children axis. A match is *total relaxed* if there is one or more relaxed match along any one of the atomic valued axes or a total relaxed match along the children axis. A *partial exact* match implies an

exact match along all atomic valued axis and a partial exact match along the children axis. Similarly, a *partial relaxed* match is a relaxed match along one or more atomic valued axis and/or a partial relaxed match along the children axis.

Consider once again the PO and the Purchase Order schemas given in Figures 1 and 2 respectively. Here the leaf nodes (children of `Lines`), `Quantity` and `Unit Of Measure` have a *relaxed* match with `Qty` and `UOM` (children of `Items`) respectively, while the child `Item` of `Lines` has an exact match with the child `Item#` of the element `Items`. Thus, the QoM of the match between `Lines` and `Items` is said to be *total relaxed* along the children axis. The elements `Lines` and `Items` have a *relaxed* match along the label and level axis (they are at different levels in the schema tree), and an *exact* match along the properties axis. Combining the QoM along all axes, we state that there is a *total relaxed* match along between the elements `Lines` and `Items`.

**Tree Match.** In an XML Schema, both *complex* elements, wherein an element may nest other elements and have attributes, and schema element can be defined analogously. The match between such complex elements, termed *tree match*, is a special case of the sub-tree match and can be classified similar to the sub-tree match.

Consider again the two schemas given in Figure 1 and Figure 2, and the match between the two root elements PO and Purchase Order respectively. The two root elements are said to have a relaxed match along the label and properties axis. Next let us consider the children axis. The PO root has three children, `OrderNo`, `PurchaseInfo` and `PurchaseDate`, while the Purchase Order has five children namely, `OrderNo`, `BillTo`, `ShipTo`, `Items` and `Date`. There is an exact match between the leaf children nodes labeled `OrderNo`, and a relaxed match between the children nodes `PurchaseDate` and `Date`.

Now consider the non-leaf child node `PurchaseInfo`. We match the sub-tree rooted at `PurchaseInfo` with all sub-trees in the Purchase Order schema. Comparing `PurchaseInfo` with the node Purchase Order in the Purchase Order schema, we have the following breakdown of QoM. The two nodes `PurchaseInfo`

and Purchase Order have a relaxed match along the label and properties axes. Now consider the QoM along the children axis. The children (leaf nodes) BillingAddr and ShippingAddr have a relaxed match with the leaf nodes BillTo and ShipTo in the Purchase Order schema. We have shown in Section 2.2, that the sub-trees rooted at nodes Lines and Items, i.e., the two non-leaf nodes Lines and Items have a *total relaxed* match. Thus, the two nodes PurchaseInfo and Purchase Order have a *total relaxed* match along the children axis. There is no level match between the two nodes. Hence the node PurchaseInfo has a *total relaxed* match with the node Purchase Order.

Based on the match of the node PurchaseInfo, we say that the node PO, the root node, has a *total relaxed* match along the children axis. Given the height difference between the schema trees, we state that there is no level match between the roots PO and Purchase Order. Combining the matches along the different axes, the QoM for the match between the PO and Purchase root nodes is said to be *total relaxed*.

### 3 Match Model: Defining a Quantitative Measure of a Match

Based on the qualitative analysis presented in Section 2.1, it can be observed that there is a “goodness” hierarchy between the different QoM classifications. For example, a *total exact* is clearly a better match than a *total relaxed* or the other classifications. However, the distinctions between the qualitative QoM classifications of *partial exact* and *total relaxed*, or of two *partial* matches are not immediately apparent. To further classify this distinction between two matches, we now provide a *weight based match model* that *quantitatively* categorizes and ranks the QoM.

As stated in Section 2.1, a match is classified based on the QoM along four axes: label, properties, children and level. However, intuitively we know that not all axes have an equal weight in determining the final match. For example, the match along the children axis may be of a higher significant value during the comparison of two nodes, than say the match between their properties or even the match of their levels. In keeping with this intuition, we assign

weights to the QoM of each individual axis to factor in their importance in deciding the overall match of the node. The QoM of a node is thus given as:

$$QoM(n_1, n_2) = W_L * QoM_L + W_P * QoM_P + W_H * QoM_H + W_C * QoM_C \quad (1)$$

where  $QoM_L$  is the QoM along the label axis,  $QoM_P$  the QoM along the properties axis,  $QoM_H$  the QoM along the level axis and  $QoM_C$ , the QoM along the children axis. The weights  $W_L$ ,  $W_P$ ,  $W_H$ , and  $W_C$  are the respective weights for the label, properties, level and children axis. The highest match classification, *total exact* will always result in a  $QoM(n_1, n_2) = 1$ . In the following sections, we now consider the quantitative measure of the QoM for a leaf match, a subtree match and a tree match.

**Leaf Match.** A leaf match is determined by comparing the values along the label and the properties axes, and is qualitatively given as either exact or relaxed. We ignore the structural and level<sup>1</sup> axes in this comparison. Thus, the QoM for two leaf nodes  $n_1$  and  $n_2$  is given as:

$$QoM(N_s, N_t) = W_L * QoM_L + W_P * QoM_P + C \quad (2)$$

where  $C$  is a constant to account for the fact that leaf nodes have exact match by default on children and level axes.

**Subtree Match.** A non-leaf node is compared along all four axes, label, properties, structure (children) and level. A match along the label and properties axes,  $QoM_L$  and  $QoM_P$  respectively, are straightforward and are calculated as described for leaf nodes. A match along the children axis is calculated based on (1) the *subtree weight*; and (2) the *cardinality ratio*. The *subtree weight*, denoted as  $\mathcal{R}_w$ , is defined as the normalized sum of the quality of the children matches, and is given as:

$$\mathcal{R}_w(N_s, N_t) = \frac{\sum QoM(n_s, n_t)}{|N_s|} \quad (3)$$

<sup>1</sup>Note that the QoM model provides the flexibility for comparing a leaf node with a non-leaf node by simply altering the level axis QoM.

where  $N_s$  and  $N_t$  are the source and the target nodes that are being compared; and  $n_s$  and  $n_t$  represent the children nodes of the node  $N_s$  and  $N_t$  respectively whose QoM value is above the threshold value.

The *cardinality ratio*, denoted as  $\mathcal{R}_s$ , is defined as the ratio of the number of children matches to the total number of children of the source node  $N_s$ , and is given as:

$$\mathcal{R}_s(N_s, N_t) = \frac{|N_s^c|}{|N_s|} \quad (4)$$

where  $|N_s^c|$  is the number of children matches and  $|N_s|$  is the total number of children for the node  $N_s$ . Thus the normalized QoM for a node  $N_s$  along the children axis is given as:

$$QoM_C(N_s, N_t) = \frac{\mathcal{R}_w(N_s, N_t) + \mathcal{R}_s(N_s, N_t)}{2} \quad (5)$$

A match along the level axis, denoted as  $QoM_H$ , is simply given as 1 if there is a level match and 0 otherwise. Based on the QoM along the different axes, we now calculate the QoM for a non-leaf node, denoted as  $QoM(N_s, N_t)$ , as

$$QoM(n_1, n_2) = W_L * QoM_L + W_P * QoM_P + W_H * QoM_H + W_C * QoM_C \quad (6)$$

**Tree Match.** QoM of the root node gives the total match weight of the schema itself and can be calculated in a manner similar to the subtree match.

## 4 The QMatch Algorithm

Based on our qualitative analysis, i.e., our match taxonomy presented in Section 2, and our cost model as presented in Section 3, we now describe a *hybrid* schema match algorithm to match two schema trees. Figure 3 gives the pseudo-code for the *hybrid* match algorithm. The algorithm presents a recursive depth first search algorithm that uniquely combines together traditional linguistic and structural algorithms, as well as algorithms that harness domain type and atomic properties for deciding on a match. The

match algorithm begins with the matching of the roots of the two input schema trees. The QoM of the two root nodes is calculated based on the QoM function given in Equation 7, where  $QoM_L$  is the QoM along the label axis,  $QoM_P$  the QoM along the properties axis,  $QoM_H$  the QoM along the level axis and  $QoM_C$ , the QoM along the children axis. The highest match classification for each of these axes will always result in a  $QoM = 1$ .

$$QoM = QoM_L + QoM_P + QoM_H + QoM_C \quad (7)$$

QMatch first recursively calculates the  $QoM_C$ , that is, the quality of match of all children nodes, followed by the match along the atomic-valued axes, that is the label axis ( $QoM_L$ ), the level axis ( $QoM_H$ ), and the properties axis ( $QoM_P$ ). The final QoM is determined as per Equation 7, and the total match value (QoM) for the entire source schema tree with respect to the target schema tree is presented to the user. The running time of the algorithm lies in **O(nm)**.

## 5 Experimental Evaluation

We conducted several experiments to evaluate both the performance and the accuracy of the **QMatch** algorithm with respect to linguistic and structural algorithms. For the benefit of this study, we developed linguistic and structural algorithms based on the algorithms presented as part of CUPID [12]. We use the same linguistic and structural algorithms internally within the **QMatch** algorithm.

**Experimental Setup and Methodology.** The structural, linguistic and hybrid **QMatch** algorithms were all implemented in Java (SDK 2.0) and evaluated on Dell 2.0 GHz Pentium 4 workstation with 512 Mb of RAM. All experiments were based on XML schemas taken from three different domains namely, Inventory, Books and Protein, as well as schemas from XML benchmark, XBench [16]. Table 1 summarizes the schemas based on the total number of their elements and their maximum depth. Full listing of the schemas can be found in [8].

	PO1	PO2	Article	Book	DCMDItem	DCMDOrd	PIR	PDB
# Eluates.	10	9	18	6	38	53	231	3753
Max. Depth	3	3	3	2	2	3	6	7

**Table 1. Characteristics of the Test Schemas.**

```

double TreeMatch (Tree: t1, Tree: t2)
{
  Node: rs, rt
  rs ← getRoot( ts )
  rt ← getRoot( tt )

  STs ← get all subtrees rooted at children nodes of rs
  STt ← get all subtrees rooted at children nodes of rt

  QoMsum ← 0 // QoM summation for current node
  -Nsc ← 0 // Number of matching children
  for all ti in STs {
    for all tj in STt {
      // calculate QoM for all matching children
      QoMc ← TreeMatch(ti, Tj)
      if QoMc ≥ threshold value
        QoMsum ← QoMc + QoMsum
        -Nsc ← -Nsc + 1
    }

    Rw ←  $\frac{QoM_{sum}}{|t_i.getRoot()|}$ 
    Rs ←  $\frac{|N_s^c|}{|t_i.getRoot()|}$ 
    // calculate the QoM for the node
    QoMC =  $\frac{R_w + R_s}{2}$ 
    QoML ← linguisticMatch (rs.label, rt.label)
    QoMP ← propertyMatch (rs.properties, rt.properties)
    if (rs.getLevel() == rt.getLevel())
      QoMH ← 1.0
    else QoMH ← 0.0

    // QoM for the parent node
    QoM = WL*QoML + WP*QoMP + WH*QoMH +
    WC*QoMC
    return QoM
  }
}

```

**Figure 3. QMatch - The Hybrid Match Algorithm.**

## 5.1 Preliminary Results

**Determining Weights of the Different Axes.** The accuracy of the **QMatch** algorithm is dependent on the weights of the different axes – an integral part of the weight-based match model defined in Section ???. To determine optimal

Label	Properties	Level	Children
0.3	0.2	0.1	0.4

**Table 2. Weight for the Different Axes.**

values for the different weights, we conducted a set of experiments that computed the match values for two randomly selected schemas, for different weight values. The overall match values obtained via the **QMatch** algorithm for the different weight values were compared against expected match values that were manually determined prior to the experiments. We then repeated the experiments for different pairs of schemas from different domains to determine if the selected weight values would hold or would need to be adjusted. We found that a weight of 0.25 to 0.4 was ideal for the label axis, a weight value in the range of 0.1 to 0.2 was ideal for the properties and level axes, while the children axis tended to be the most significant weight and ranged between 0.3 and 0.5. Table 2 shows the weights that were chosen for the **QMatch** algorithm.

**Algorithm Performance.** The comparison measurements, experiments that compared the runtime performance of the three algorithms, matched schemas taken from the same domain. A second set of experiments was also conducted to measure and compare overall quality of the three algorithms.

We considered a subset of schemas given in the Table 1 and recorded the running time for all three algorithms (linguistic, structural and **QMatch**) for these schemas. Figure 4 gives the overall performance comparison for the three match algorithms, linguistic, structural and **QMatch**. As can be noted the runtime performance of the **QMatch** algorithm is worse than that of the linguistic and structural algorithms. This is as expected as the hybrid **QMatch** algorithm combines both linguistic and structural algorithms to calculate the QoM between the schemas.

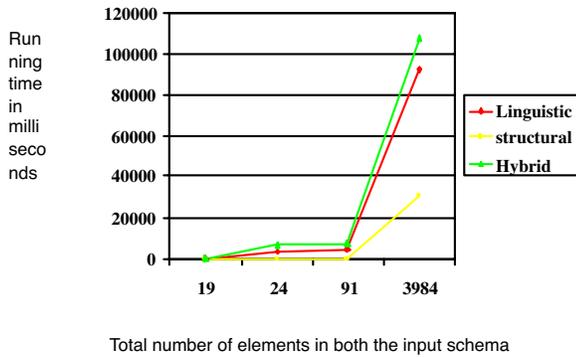


Figure 4. Overall Performance of match algorithms

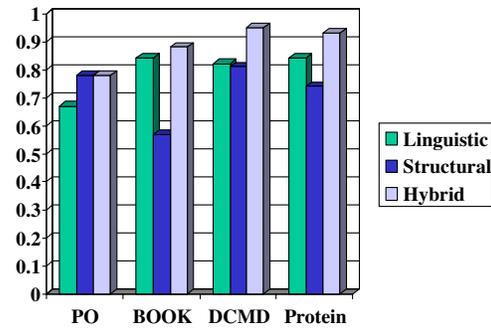


Figure 5. Comparison of the Overall Measure of Match Quality for the Linguistic, Structural and QMatch Algorithms.

**Algorithm Quality.** To evaluate the quality of our approach, we compared the manually determined real matches ( $R$ ) for a given match task with the matches  $P$  returned by the match algorithm. We determined the true positives, i.e. correctly identified matches,  $I$ , the false positives, i.e. false matches,  $F = P \setminus I$ , and the false negatives, i.e. missed matches,  $M = R \setminus I$ . Based on the cardinalities of these sets, the *Precision* and *Recall* of the match algorithm were computed.

- $Precision = \frac{|I|}{|P|} = \frac{|I|}{|I|+|F|}$  estimates the reliability of the match predictions
- $Recall = \frac{|I|}{|R|}$  specifies the share of real matches that is discovered by the algorithm
- $Overall = 1 - \frac{|F|+|M|}{|R|} = \frac{|I|-|F|}{|R|} = Recall * (2 - \frac{1}{Precision})$  represents a combined measure of match quality, taking into account the post-match effort needed for both removing false matches and adding missed matches.

We evaluated the overall measure of match quality for the three algorithms and compared them against each other. We considered schemas from different domains, and compared PO1 and PO2, Article and Book, DCMDItem and DCMDOrd and finally two protein schemas, PIR and PDB. Figure 5 shows the comparison of these algorithms in terms of the measured quality.

Figure 6 shows the comparison between the total number of manual matches and the total number of matches found

by each of these algorithms for the different input schemas. The graph does not depict a comparison for protein schemas as it has thousands of elements, and it is nearly impossible to accurately determine the matches manually. However, based on the trends exhibited by the match algorithms for the other domains, it is possible to extrapolate that the algorithms will behave in the same lines for these schemas too.

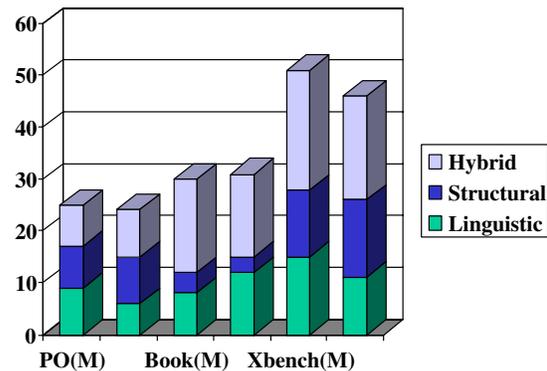


Figure 6. Comparison of Manual ( $R$ ) vs the Matches ( $P$ ) Found by the Three Algorithms.

In general we found that **QMatch** algorithm did better both in terms of the total number of matches found, as well as in terms of the accuracy of the discovered matches. This was consistent with our initial hypothesis and held for all

cases where the linguistic and structural algorithms returned matches in the same ballpark quality. Moreover if linguistic and structural algorithms were in the same quality range, we found an increased accuracy of QoM as determined by the **QMatch** algorithm. However, we did notice a tendency in the **QMatch** algorithm to average the match quality when the match values obtained from the linguistic and structural algorithms are individually on two ends of the quality spectrum - low vs high. Specifically, if the two input schemas are linguistically disparate and structurally identical as shown in Figures 7 and 8 or vice versa, we observed that the accuracy results of the **QMatch** algorithm gravitated towards the higher individual algorithm (linguistic or structural) values.

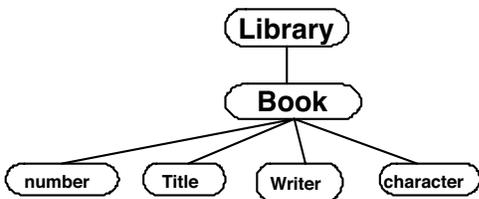


Figure 7. Library schema

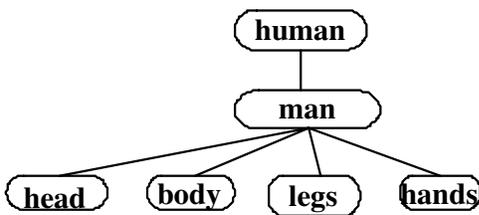


Figure 8. Human schema

Figure 9 shows the total QoM values for the schemas given in Figures 7 and 8. While intuitively this is in accordance with the fact that **QMatch** combines linguistic and structural algorithms, it also suggests that for such extreme cases some tuning of the match weights may be required to provide a more average match quality.

## 6 Related Work

Many schema match algorithms [3, 4, 2, 6, 12, 5, 13, 14, 15] have been proposed in literature to address the problem of schema integration. These algorithms typically rely

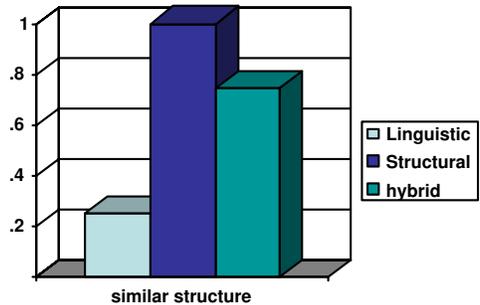


Figure 9. Overall Measure of Match Quality for Linguistic, Structural and QMatch Algorithms for Two Structurally Identical but Linguistically Different Schemas

on two primary factors to detect similarities between the schema entities: the *label* and the *structure* of the involved entities. Integration systems [4, 2, 3, 6, 12, 5] typically provide either individual algorithms based on these two basic criteria, i.e., linguistic or structural matching, or some combination of the two [12, 7]. For example, Neirman et al [15] have proposed a structure-based similarity algorithm that determines a match between XML documents based on measuring the edit distance for the rooted XML trees. Madhavan et al., on the other hand, have proposed CUPID [12] that combines linguistic and structural matching to establish correspondences between the schema entities. **TranScm** [13], on the other hand, performs schema matching at the granularity of elements using a set of predefined rules to match node by node on the graph representation of the input schemas. **SKAT** [14] focuses on identifying the articulation over two ontologies by comparing input schemas based on their ontological knowledge sources and user defined rules which are utilized to increase precision. Two nodes are once again matched based on their labels. Structural similarity is established based on the similarity of their

hierarchical structure.

Other systems such as **LSD** [1] and **SemInt** [11] use machine learning and neural networks frameworks respectively as an approach for combining different match techniques and user feedback to ultimately improve the accuracy of schema matching. LSD employs and extends current machine learning techniques to semi-automatically find mappings between input schemas. LSD uses user-supplied semantic mappings for a small set of data sources together with the sources to train a set of learners. Each learner exploits different information in the input schemas. Once they are trained, LSD finds semantic mappings for a new schema by applying the learners and then combining their predictions using a meta-learner. **SemInt** trains neural networks to find matches between two input schemas. **SemInt** provides a match procedure using a classifier to categorize attributes according to their field specifications and data values and, then trains a neural network to recognize similar attributes. It does not support any linguistic matching.

While the basis of our work is still linguistic and structural algorithms, we differ in how these these are combined. In our work, we focus on the disciplined combination of all aspects of the semantic and structural information captured in XML schemas targeted towards providing better accuracy and value.

## 7 Conclusions

In spite of the many matching techniques that have been explored in the literature, schema matching still remains a challenging area of work. To address this challenge, in this paper we have presented a hybrid schema matching algorithm **QMatch** that is specifically targeted towards the matching of XML schemas. It is based on the measurement of a unique quality of match metric, QoM, which provides not only an effective basis for the development of a new schema match algorithm, but also a useful tool for tuning existing schema match algorithms to output at desired levels of matching. Preliminary results show that in the average case **QMatch** outperforms the linguistic and structural algorithms both in terms of the accuracy of the matches as well as in terms of the total matches discovered. It should be noted that the linguistic and structural algorithms used here

can be easily replaced by other perhaps better performing linguistic and structural algorithms. Our current ongoing work is focused on evaluating the quality of match and the performance of **QMatch** with other hybrid and composite algorithms such as CUPID [12] and COMA [5].

## References

- [1] D. AnHai, P. Domingos, and A. Haley. Reconciling Schemas of Disparate Data Sources: A Machine-Learning approach. In *SIGMOD*, 2001.
- [2] S. Bergamaschi, S. Castano, M. Vincini, and D. Benevenuto. Semantic integration of heterogeneous information sources. *Data and Knowledge Engineering*, 36(3):215–249, 2001.
- [3] J. Berlin and A. Motro. AutoPlex: Automated Discovery of Content for Virtual Databases. In *CoopIS*, pages 108–122, 2001.
- [4] M. Bright, A. Hurson, and S. H. Pakzad. Automated Resolution of Semantic Heterogeneity in Multidatabases. *TODS*, 19(2):212–253, 1994.
- [5] H. H. Do and E. Rahm. COMA - A System for Flexible Combination of Schema Matching Approaches. In *vldb*, 2002.
- [6] L. Haas, R. Miller, B. Niswonger, M. Roth, P. Schwarz, and E. Wimmers. Transforming Heterogeneous Data with Database Middleware: Beyond Integration. *IEEE Data Engineering Bulletin*, 22(1):31–36, 1999.
- [7] L. Haas, R. Miller, B. Niswonger, M. Roth, P. Schwarz, and E. Wimmers. Transforming Heterogeneous Data with Database Middleware: Beyond Integration. *IEEE Data Engineering Bulletin*, 22(1):31–36, 1999.
- [8] V. Hegde. Qom: A taxonomy based approach to schema matching. In *Master's Thesis, UMass-Lowell*, April 2004.
- [9] G. Inc. Google: A Web Search Engine. [www.google.com/](http://www.google.com/).
- [10] L. Inc. Hotbot: A Web Search Engine. [www.hotbot.com/](http://www.hotbot.com/).
- [11] W.-S. Li and C. Clifton. Semantic Integration in Heterogeneous Databases Using Neural Networks. In *vldb*, 1994.
- [12] J. Madhavan, P. Bernstein, and E. Rahm. Generic Schema Matching with Cupid. In *vldb*, pages 49–58, 2001.
- [13] T. Milo and Z. Sagit. Using Schema Matching to Simplify Heterogeneous Data Translation. In *vldb*, 1998.
- [14] P. Mitra, G. Wiederhold, and J. Jannink. Semi-automatic Integration of Knowledge Sources. In *fusion*, 1999.
- [15] A. Neirman and H. Jagadish. Evaluating Structural Similarity in XML Documents. In *webdb*, 2002.
- [16] U. of Waterloo. XBench - A Family of Benchmarks for XML DBMSs. <http://db.uwaterloo.ca/dbms/projects/xbench/>.
- [17] N. Tansalarak and K. Claypool. QoM:Qualitative and Quantitative Schema Match Measure. In *Conference on Conceptual Modeling (ER)*, 2003.
- [18] W3C. XQuery 1.0: An XML Query Language. <http://www.w3.org/TR/xquery/>, 2003.