Abstract—We propose a novel solution for semantic-based XML schema matching, taking a mathematical programming approach. This method identifies the globally optimal solution for the problem of matching two XML schemas by reducing the tree-to-tree matching problem to simpler problems of path-to-path, node-to-node, and word-to-word matching. We formulate these matching problems as maximum-weighted bipartite graph matching problems with different constraints, which are solved by different mathematical programming techniques, including integer programming and dynamic programming. Solutions to simpler problems provide weights for the next stage until the optimal tree-to-tree matching solution is obtained. The effectiveness of this approach has been verified and demonstrated by computer experiments.

Keywords—E-business; XML schema matching; optimization; maximum-weighted bipartite graph; semantic similarity; integer programming; dynamic programming

I. INTRODUCTION

XML [1] and XML schemas [2] have been widely used in the e-Business transactions among enterprises that exchange business documents with their partners (e.g., suppliers and customers in the supply chain) [3,4,5]. Many enterprises and organizations have defined their own XML schemas to describe the structure and content of their business documents (i.e., XML instances) to be used in the transactions. Many organizations have also published standard XML schemas to be shared in the transactions within specific industry domains [6,7,8] (e.g., e-manufacturing, e-government, and e-health industries).

The popularity of XML leads to an integration problem as different enterprises or organizations often choose different XML representations for the same or similar concepts [4,5]. One of the most critical steps to achieving the seamless exchange of information between heterogeneous e-Business systems is schema matching, which is known to be costly and error-prone [9,10]. Schema matching takes as input two schemas, each consisting of a set of discrete components (elements or attributes), and determines as output the relationships between these components [11].

XML schemas or instances are typically viewed as labeled trees (i.e., rooted acyclic graphs) where each node represents a component named by a label of English word or concatenation of words or their abbreviations. In this paper, we focus on one type of schema mapping, namely, matching between all atomic components (i.e., the leaf nodes) between two schemas or instances based on their semantics (meaning of nodes). Also, we only consider one-to-one matchings of 1:1 cardinality (e.g., any atomic component in the source schema can match no more than one atomic component in the target schema). Matching between those atomic components helps to identify how a certain value of one XML instance can be transformed to certain value of the other for successful exchange of information.

We propose new innovative techniques to address two challenging problems in this type of schema matching. First, due to synonyms (different words meaning the same thing) and multi-senses (one word having different meanings in different context) found in natural languages, the meaning of an atomic component cannot be determined solely by the words in its label. The semantic ambiguity can be reduced by contextual information such as the labels of its neighboring nodes. In this paper, we concentrate on one type of context for an atomic component: the nodes along the path from the root to the leaf of the schema tree.

Second, it is difficult to correctly identify the best set of matching pairs for all atomic components between two trees. This is because a leaf in one tree may match more than one leaf in the other tree (with different semantic similarities) and locally identified best-matching pairs do not necessarily form the globally optimal set. We propose to use mathematical programming techniques to solve this combinatorial optimization problem. To further reduce the computational complexity, we propose to decompose the global problem into simpler matching problems such as path-to-path, node-to-node, and word-to-word matching. We formulate the sequence of matching problems as maximum-weighted bipartite matching problems with different sets of constraints. We solve them by different mathematical programming techniques, including integer programming [12] and dynamic programming [13]. Solutions to simpler problems provide weights for the next stage until the optimal tree-to-tree matching is obtained.

The remainder of the paper is organized as follows. Section II provides a brief survey of the related works. The detailed algorithm of the proposed approach is described in Sections III. Section IV reports the experiments and results. Section V concludes with the directions for future research.

II. RELATED WORKS

Many schema matching methods have been proposed [10,11]. Typically, these methods first attempt to identify
semantic relationships between the elements of two schemas. Based on the granularity of the matching, these methods can be separated into two classes: element-level and structure-level [11]. The element-level approaches determine the matching elements in the target schema for each element of the source schema. The structure-level approaches refer to matching combinations of elements that appear together in a structure.

For the element-level matching, a string-based similarity metric is commonly used to analyze the linguistic context of names and name descriptions of schema elements. There are a number of such metrics including Hamming distance [14], Jaccard similarity coefficient [15], Cosine coefficient [16], and n-gram [17]. They can be enhanced using natural language preprocessing techniques for the input string, such as tokenization, lemmatization, and elimination [11].

To further enhance the string-based metrics, corpus resources can be utilized for more accurate and less ambiguous results. One of the important resources is the lexical taxonomy among the words (e.g., parents, children, ancestor, and descendant relationships). Common knowledge corpora, such as WordNet [18] and domain-specific corpora can be used to determine the meaning of the words [19,20].

A corpus also provides statistical information related to the importance of words. The differences in importance of individual entities and their relationships affect the semantic similarity measurement. The information content (IC)-based metric was proposed to utilize this statistical information [21,22,23]. This approach measures the similarity between two entities - two words, two objects, or two structures - \( A \) and \( B \) based on how much information is needed to describe \( \text{common}(A, B) \), the commonality between them. Examples of commonality include the features or hypernyms the two words share. Applying this approach to tree-like IS-A taxonomies [22], one can measure the similarity between \( A \) and \( B \) as

\[
\text{Sim}_{ic}(A, B) = \frac{2 \cdot \log P(C)}{\log P(A) + \log P(B)},
\]

where \( C \) is the most specific subsumer of \( A \) and \( B \) with the smallest prior probability and the probabilities can be obtained according to the frequencies in a corpus.

For structure-level matching, a variety of graph-based metrics have been proposed [10,11]. Typically, these metrics quantify the commonality between components by taking into account the lexical and structural similarities of sub-components using ancestors and descendents including leaf components. Because most schemas can be viewed as labeled trees, many matching algorithms have been developed based on either top-down or bottom-up traversal techniques [10].

As an example of the top-down approach, TransScm [24] provides a schema-based matching for data translation and conversion based on the syntactic analysis of the structures. Tess [25] is another example of a top-down algorithm, which deals with schema evolution. Tess takes definitions of the old and new types and identifies pairs of types as matching candidates. It then recursively tries to match their substructure in a top-down fashion.

Alternatively, the bottom-up approach compares all combinations of the elements and finds matches at a given level, even if the intermediate and higher level structures differ considerably. Similarity flooding (SF) [26] provides bottom-up matching based on similarity propagation. Another effective bottom-up method, called S-Match [27], decomposes the tree-matching problem into a set of node-matching problems. Each node-matching problem is translated into a propositional formula. This transforms the matching problem into a propositional unsatisfiability problem, which can then be resolved efficiently using the state-of-the-art propositional satisfiability deciders.

Cupid [28] provides a composite matching approach that combines element- and structure-level matching approaches. It computes the similarity using domain-specific thesauri as the linguistic resources; and, it traverses the tree in a combined bottom-up and top-down manner.

Top-down approaches are less expensive but can be misled if the top-level schema structures are very different [10]. On the other hand, bottom-up approaches take a more comprehensive view [11]. However, they identify optimal matches using ad hoc approaches that have poor theoretical foundations.

III. ALGORITHM

We propose a novel schema-based matching algorithm to solve the combinatorial matching optimization problem of matching atomic components between two schemas. Our approach synthesizes the global optimal set of pair-wise matchings between atomic components in a principled manner by mathematical programming.

A. Algorithm overview

The matching algorithm takes as input two schemas and identifies the set of matching pairs of all atomic components with the highest semantic similarity among all possible sets of pair-wise matchings. Fig. 1 illustrates the matching process of our approach for two input schemas, \( S1 \) and \( S2 \).

The algorithm breaks the complex combinatorial optimization problem into four matching stages: tree-to-tree, path-to-path, node-to-node, and word-to-word matchings. As can be seen in Fig. 2, each stage works on a bipartite graph, consisting of two sets of vertices and a weight matrix with the objective of finding the 1:1 matching between vertices in one set to the other with the highest overall weight. Therefore, we formulate these sub-problems as maximum-weighted bipartite graph matching problems [29].
ExCEPT FOR THE word-to-word matching stage, the weight matrix for each stage is a similarity matrix calculated by the previous stage. For example, the similarity matrix for tree-to-tree matching stage is provided by path-to-path matching stage. The word-to-word matching stage uses WordNet to compute the semantic similarity between two words by identifying the optimal matching pairs for their respective senses.

Graph with two sets of vertices, $U = \{u_1, u_2, \cdots, u_n\}$ and $V = \{v_1, v_2, \cdots, v_m\}$, and the edge set $E$. Edge $e_{ij}$ in the graph connects the vertices $u_i$ and $v_j$ whose weight $w_{ij}$ is given in the weight matrix $W$. Vertices of the same set are not connected.

A matching $M$ of graph $G$ is a subset of $E$ such that no two edges in $M$ share a common vertex. In other words, $M$ consists of a set of pair-wise matchings of 1:1 cardinality. The maximum-weighted bipartite matching is a matching whose sum of the weights of the edges is the highest among all possible sets of pair-wise matchings. The optimal matching $M$ can be found by integer programming defined below:

\[
\text{Maximize: } \sum_{e_{ij} \in E} W_{ij} x_{ij} \tag{2}
\]

subject to:

\[
\sum_{j=1}^{m} x_{ij} = 1, \forall j = 1, \cdots, n, \sum_{i=1}^{n} x_{ij} = 1, \forall i = 1, \cdots, m,
\]

\[
x_{ij} \in \{0,1\}, \text{ where } x_{ij} = 1 \text{ if } e_{ij} \in M \text{ and } 0 \text{ otherwise.}
\]

Because integer programming is typically NP-hard (i.e., harder than a nondeterministic polynomial-time problem and for worst case with running time exponential to the problem size) [32], we approximate it by a simple greedy algorithm as follows:

\[
\text{algorithm } MBM\text{-greedy}(U,V,W) \\
\text{sort } W; \\
\text{while } (|U|>0 \text{ and } |V|>0) \text{ do:} \\
\hspace{1em} \text{Choose vertices } u \text{ and } v \text{ connected with an edge } e \text{ that has the highest weight } w \text{ in the weight matrix } W; \\
\hspace{1em} \text{if edges in } M \text{ share either } u \text{ or } v \text{ then} \\
\hspace{2em} M := M + \{e\}, U := U - \{u\}, V := V - \{v\}; \\
\hspace{2em} w_{\text{new}} := \text{wsum} + w; \\
\hspace{2em} \text{wsum} := 2 \times \text{wsum} / (m+n); \\
\hspace{2em} \text{return } (M, \text{warg}); \\
\end{algorithm}

Figure 3. Greedy algorithm for maximum-weighted bipartite matching.
The greedy algorithm simply sorts the weight matrix $W$ in ascending order and at each iteration chooses an edge with the highest weight. The initial weight matrix $W$ is calculated by the previous matching stage. The chosen edge will be the matching candidate if it shares no vertex with edges in $M$. This process is repeated until there is no vertex to be matched in either $U$ or $V$. The algorithm returns the optimal matching set $M$ and the average weight of all edges in $M$ as the measure of similarity between $U$ and $V$. In this greedy algorithm, the most expensive step is the sorting of the weight matrix $W$ of size $m \times n$. We use a quicksort algorithm that takes $O(k \log(k))$ to sort $k$ items. Thus, the complexity of this greedy algorithm is $O(mn \log(mn))$.

C. Ordered maximum-weighted bipartite matching algorithm

Some have suggested using the longest common sequence (LCS) to address the ordering criterion in the path-to-path matching [30,34,35]. However, none of the suggestions utilizes the semantic similarities of the nodes on the two paths. To consider semantic similarities of the nodes, we have developed the ordered maximum-weighted bipartite matching algorithm based on dynamic programming.

Let $G$ be a weighted bipartite graph with two ordered sets of vertices $U = \{u_1, u_2, \ldots, u_n\}$ and $V = \{v_1, v_2, \ldots, v_n\}$, and the edge set $E$. The core algorithm, $OMBM(U, V)$, finds the optimal matching $M$ between $U$ and $V$ by recursively partitioning it into smaller sub-problems until the solution becomes trivial.

For a sequence $S = s_1, s_2, \ldots, s_k$, a sequence shortened from the end is denoted $S^{k\rightarrow} = s_1, s_2, \ldots, s_k$, where $k \leq d$. We call $S_k$ the prefix of $S$. The prefixes of $U$ are $U_1, U_2, \ldots, U_m$, and the prefixes of $V$ are $V_1, V_2, \ldots, V_n$. Let $OMBM(U, V)$ be the function that finds the optimal matching of prefixes $U_i$ and $V_j$. This problem can be reduced to three alternative simpler sub-problems with shortened prefixes and returns the one with maximum sum of weights:

1) $u_i$ and $v_j$ match each other. Then, the optimal matching for $U_i$ and $V_j$ can be formed by attaching edge $e_{ij}$ to the optimal matching of two shortened sequences $U_{i+1}$ and $V_{j-1}$, denoted $OMBM(U_{i+1}, V_{j-1}, e_{ij})$.

2) $u_i$ and $v_j$ do not match each other. Then, either of them can be removed to shorten one of the matching sequences and $OMBM(U_i, V_j)$ is reduced to either $OMBM(U_{i+1}, V_j)$ or $OMBM(U_i, V_{j+1})$.

Thus $OMBM(U, V)$ can be computed by the following recursive function:

$$OMBM(U_i, V_j) = \begin{cases} \emptyset & \text{if } i = 0 \text{ or } j = 0 \\ \max(OMBM(U_{i+1}, V_j), OMBM(U_i, V_{j+1}), OMBM(U_{i+1}, V_{j+1}), e_{ij}) & \text{otherwise} \end{cases},$$

where the function $\max(t)$ returns the optimal matching among the three matchings from the sub-problems; it returns empty if either $U_i$ or $V_j$ is reduced to null ($i = 0$ or $j = 0$).

The optimal matching $M$ of two sets of order vertices $U$ and $V$, $|U| = n$, $|V| = m$, is then computed as:

$$M = OMBM(U, V).$$

The similarity score of $M$, denoted $Sim_{OMBM}$, is the average weights of all edges in $M$:

$$Sim_{OMBM} = \frac{\sum_{e_{ij} \in M} w_{ij}}{m + n}. \tag{5}$$

To efficiently execute the algorithm, we use a bottom-up approach [13]. The algorithm is as follows:

```
algorithm OMBM-arrays (U,V,W)
for i from 1 to n
    for j from 1 to m
        warg := 2*max(A[i,j], A[i-1,j], A[i,j-1]) /
    return [A, warg];
```

Figure 4. Bottom-up dynamic programming algorithm for ordered maximum-weighted bipartite matching.

This algorithm starts from the simplest matching between $U_1$ and $V_1$ and continues to more complex matching problems. The calculated similarity scores for the optimal matchings (average weights by Eq. (5)) are stored in table $A[i,j]$ in Fig. 4 for future use to avoid repeated calculations of smaller problems. The weights of $W$ are calculated by the previous matching stage (i.e., node-to-node matching stage). The complexity is only $O(mn)$, i.e., linear to the size of table $A$.

The following algorithm deals with a simple matter of finding the matching between components $U$ and $V$ that is identified by bottom-up dynamic programming algorithm of Fig. 5.

```
algorithm OMBM (U,V,W)
OMBM-arrays (U,V,W);
    i := m; j := n;
    while i > 0 and j > 0
        if $A[i,j]$ equal to $A[i-1,j]$ then
            elseif $A[i,j]$ equal to $A[i,j-1]$ then
                else $M := M+\{e_{ij}\}$, $i--$, $j--$;
        return [M, warg];
```

Figure 5. Dynamic programming algorithm for ordered maximum-weighted bipartite matching.

Algorithm $OMBM$ is further enhanced by considering the differences in importance for the individual components measured by their IC values. We collect each component’s frequency-of-occurrence in the schemas and compute the IC by Eq. (1). Fig. 6 shows the modified algorithm.

Algorithm $OMBM$-IC gives more weights to higher-level components because lower-level components are typically generic entities that appear widely as the descendants of the higher-level components. In addition, it also considers the differences in importance of components at the same level. The complexity of this algorithm is still $O(mn)$.
Algorithm OMWM-IC \((U,W)\)

for \(i\) from 1 to \(m\) 
  ic-sum := ic-sum + ic\((u_i)\);
for \(j\) from 1 to \(n\) 
  ic-sum := ic-sum + ic\((w_j)\);
for \(i\) from 1 to \(m\) 
  for \(j\) from 1 to \(n\)
    ic-weight := \(W\) \((u_i, v_j)\) + ic\((u_i)\) + ic\((v_j)\); 
    \(A[i,j] := \text{maximum of } A[i-1,j], A[i,j-1], A[i-1,j-1] + \text{ic-weight};\)
    \(warg := A[m,n]\) 

The algorithm views matching two schema trees as matching two sets of atomic components with their respective path-contexts. Each path consists of a sequence of nodes along the path from the root to the leaf of the schema tree. Each node represents a component named by a label of English word or concatenation of words or their abbreviations. To compute semantics similarities between words, we analyze optimal pair-wised matchings between multiple senses of two words.

The word-to-word matching algorithm uses two semantic similarity measure functions: word-sense-sim() based on WordNet taxonomy and word-desc-sim() based on textual description. In WordNet, nouns are organized into taxonomies in which each node has a set of synonyms (a synset), each of which representing a single sense [18]. If a word has multiple senses, it will appear in multiple synsets at various locations in the taxonomy. To compute the semantic similarity between two words (two sets of senses), we use the MBM-greedy() algorithm with the input of two set of senses for words \(W_1\) and \(W_2\), respectively, and the similarities between the senses are calculated by Eq. (1).

If a word does not exist in WordNet, we extract the textual description of a given word from the internet and then use string-similarity measures, such as the cosine similarity [16], to calculate the similarity between the two textual descriptions of the two words.

IV. EXPERIMENTS AND RESULTS

We have implemented a prototype system of our approach based on Java and JWN [36] for experimental validation. In the experiments, we used five real world XML schemas for purchase orders (i.e., CIDX, Apertum, Excel, Norris, and Paragon) from [37,38]. Table I summarizes the characteristics of those XML schemas.

<table>
<thead>
<tr>
<th>Schemas</th>
<th>CIDX</th>
<th>Apertum</th>
<th>Excel</th>
<th>Norris</th>
<th>Paragon</th>
</tr>
</thead>
<tbody>
<tr>
<td>max depth</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td># nodes</td>
<td>40</td>
<td>145</td>
<td>55</td>
<td>65</td>
<td>80</td>
</tr>
<tr>
<td># leaf nodes</td>
<td>33</td>
<td>116</td>
<td>42</td>
<td>54</td>
<td>68</td>
</tr>
</tbody>
</table>

In the experiment, as suggested in [36], we compute the tree-to-tree similarity of the ten pairs of the five XML schemas. Then for each schema, we accept a matching to any of the other four if the similarity score is above a fixed threshold 0.6. To evaluate the quality of our match result, we used several performance metrics including Precision, Recall, F-measure, and Overall [39,40], against the results from manual matching [38]. These measures are then compared with the performances of other approaches for the same setting [28,38,41]. Note that the Overall measure, proposed by [38] to estimate the post-match efforts, varies in \([-1, 1]\) and other three vary in \([0,1]\).

Precision, Recall, F-measure, and Overall for our results are 0.85, 0.85, 0.85, and 0.69. To increase the precision, we used a relative threshold which is chosen as the similarity of the matching with the largest gap to the next best matching, among matching candidates with similarities ranging from 0.5 to 0.6. Fig. 8 shows the performance analysis of the matching result that our solution produced.

Figure 6. Algorithm enhanced by information contents.

Figure 7. Overall schema matching algorithm.

The experimental results show that our matching performances of Precision, Recall, F-measure, and Overall are 0.93, 0.83, 0.88, and 0.77, respectively. Comparing to the
previous results that use a fixed threshold, the Recall is slightly decreased while the Precision is significantly increased. The relative threshold also helps to increase F-measure and Overall. For comparison purposes, the average scores of performance metrics by some other methods are given in Fig. 9.

The first comparison, as illustrated in Fig. 9a, is with Thang [41] who proposed an XML schema matching solution that combines linguistic, data type, and path-context similarity measures. He also implemented the Cupid [28] algorithm for comparison purpose. We compared our result to both algorithms. In general, all performance metrics of our approach are slightly better than Thang’s and significantly better than Cupid’s.

The second comparison is with COMA [38], which used various ways for combining different matchers. Because COMA only provides performance graphs without the specific scores as shown in Fig. 9b, it is difficult to compare the performances with our result exactly. However, Fig. 9b shows that our result is, in general, at least equal to or slightly better than COMA’s results even if some of their matchers used the manual matching called SchemaM [38].

V. CONCLUSIONS

In this paper, we have described a solution to identify semantic-based optimal XML schema matching using mathematical programming. This solution identifies the optimal matching between two XML schemas on the assumption that the tree-to-tree matching problem can be globally optimized by reducing it to simpler problems, such as path-to-path, node-to-node, and word-to-word matching. We have implemented a prototype system for our solution and conducted the experiments with actual industry XML schemas. We compared our result to some other XML schema matching approaches. The results were encouraging. The average matching performances of Precision, Recall, F-measure, and Overall were 0.93, 0.83, 0.88, and 0.77, which are better than or at least equal to other approaches.

Although our approach primarily targets the XML schema matching problem, the solution can be applied to other matching problems - such as XML instance matching if the instances can be represented as labeled trees. Our solution is limited to the assumptions that only 1:1 matching cardinality is considered and that schema designers correctly use the English terminologies when labeling the elements/attributes in the schemas. These limitations call for further research. Other directions of research include methods to improve the performance by utilizing domain specific terminology and taxonomy, ontologies with formally defined concept semantics, and user feedback.

ACKNOWLEDGMENT

This work was supported in part by NIST award 70NANB9H9145.

REFERENCES


