Semantic Mapping of XML Tags using Inductive Machine Learning

Lukasz Kurgan [1,2], Waldemar Swiercz [1,2] and Krzysztof J. Cios [1,2,3,4]
[1] Department of Computer Science and Engineering, University of Colorado at Denver
[2] Department of Computer Science, University of Colorado at Boulder
[3] University of Colorado Health Sciences Center
[4] 4cData, LLC, Golden, Colorado

Abstract – *In today's data-centric world many applications rely on data that comes from multitude of different sources. To integrate that data two major operations are performed: finding semantic mapping between data sources, and transforming structure of the data sources. One of the well-established standards for storing and sharing structured and semantically described data is XML. This paper describes system called XMapper, which is used to generate semantic mapping between two XML sources that describe instances from the same domain. The described system is novel in two ways. It uses only stand-alone XML documents (without DTD or XML schema documents) to generate the mappings. It also utilizes machine learning to improve accuracy of such mappings for difficult domains. Several experiments that use artificial and real-life domains described by XML documents are used to test the proposed system. The results show that mappings generated by the XMapper are highly accurate for both types of XML sources. The generated mappings can be used by a data integration system to automatically merge content of XML data sources to provide unified information for a data processing application.*

Keywords: semantic mapping, data integration, XML, machine learning, XMapper, DataSqueezer algorithm

1. Introduction

XML (eXtensible Markup Language) is a markup language for documents that contain structured information [3]. XML is a subset of SGML [15] that uses custom-defined tags to store and describe structured, or semi-structured, data and its relationships. Structured information consists of content (numbers, character strings, images, etc.) and information of what role that content plays (e.g., a rule is build out of selectors, where a selector is a pair of attributes (name, value)). The XML tags are used to describe structure and semantic meaning of the information. One of major advantages of XML is ease of their automatic processing.

The XML technology is widely used in industry to transfer and share data. Within the computer science field, XML is widely used by database and software engineering researchers and practitioners. Although some of the computer science disciplines do not use the XML, some of them, like Data Mining and Knowledge Discovery recognize the importance of that technology [8].

1.1. Related research

Because of rapid growth of the structured data that is stored using XML documents many enterprises need to build data integration systems that provide unified access to semantically and structurally diverse information sources [14]. The data integration system has to find structural transformations and semantic mappings that result in correct merging of the information and allow user to query the so-called mediated schema [21]. It is very difficult at present to create such system because system creators have to find transformations between documents manually, on the case-by-case basis. In this paper we use XML documents as data sources.

This paper addresses problem of finding semantic mappings between structured documents within a given domain. The problem of structural transformation of XML sources was described in [30]. Majority of the approaches to semantic mapping problem concentrate on creation of mediated schema. The mediated schema is a virtual schema that captures the domain's most important aspects by creating

mappings between it and a set of data sources [27]. This paper describes system that is able to generate pairs of semantically related XML tags, which can be further mapped into a user-created mediated schema.

Most of the schema mapping systems use only structural and data type information about XML documents, which is stored in either DTD (document type definition) [13] or XML Schema [31] files.

Several mapping systems work with XML documents. The TransScm system [26] matches schema based on the structure and names of the SGML tags extracted from DTD files by using concept of labeled graphs. The LSD system [12] uses multistrategy learning by utilizing several machine learning (ML) algorithms based on the user-specified mappings to discover matching patterns. Based on these patterns, the mappings between leaf nodes in the DTD trees for two XML documents are generated. We will compare results of semantic mapping generated by our system with the results generated by the LSD system.

The reconciliation of relational schemas is approached by several systems, like Artemis [1, 4] and Clio [25, 32]. The Artemis system measures similarity of element names, data types and structures to match schemas. Clio uses SQL queries and data examples to derive and rank alternative mappings between schemas. ML based systems that use single-learner approach, like Semint [22] and Delta [11], were used to discover attribute mappings for relational databases. Delta generates a text string for every tag (attribute) that describes it and matches tags based on the similarity between these strings. The significant disadvantage of the Semint system, which uses neural networks, is low performance while working with textual information. These systems cannot handle hierarchical XML schemas.

Our system, called XMapper, also utilizes ML together with constraint analysis. The main difference between XMapper and other mapping systems is that it uses only standalone XML documents (DTD or Schema files are not used) to derive the mappings. The ML component of the XMapper system improves accuracy of mappings for difficult domains. XML data sources of such domains use data types that are either identical or very similar, and their tag names between these data sources are significantly different. In such a case correct mappings can be found only by using the ML approach. ML is used to find relationships between XML tags. Based on them corresponding tags from different XML sources are matched.

1.2. Problem Definition

The goal of the XMapper system is to provide semantic mapping that enables integration of information between two XML data sources. Two example XML sources and the mapping discovered by the XMapper system are shown in Table1.

The XMapper's assumptions are as follows:

- it generates only 1-to-1 mapping of tags between two XML documents; the same assumption is true in the LSD system
- it requires from the user to select one matching tag between two documents. This tag is used as the class label by the ML component of the XMapper system. The user-selected tag is not considered as part of the algorithm result since it is treated as a user-specified mapping.
- source XML documents are assumed not to have multiple children nodes having the same tag name. In case of such occurrence the system would only use one, structurally last instance.

2. The XMapper Algorithm

As said above the XMapper generates 1-to-1 mappings of XML tags between two source documents. The XMapper system works in two steps. First, it extracts for every tag from the two XML documents a vector of features that describes its properties. For every pair of tags, which belong to different sources, distance between their feature vectors is calculated. Next, 1-to-1 semantic mappings are generated by sequentially finding pairs of tags with the minimum distance. To account for a situation when some of the tags may not have a semantically corresponding tag in the other XML source, a threshold value is applied to the distance value. The idea of the XMapper is similar to the one used in the Delta [11] system that used text strings to describe properties of relational attributes.

In the remaining part of the paper, term "XML tag" will be substituted by attribute, since from ML point of view we consider XML files as structured datasets described by a set of attributes (tags).

Table 1. From left: XML documents for hea1 and hea2 sources (only one instance is shown), semantic mapping generated by the XMapper system

<table>
<tr>
<td>
<hea1>
  <example>
    <class>1</class>
    <Age>35</Age>
    <Sex>0</Sex>
    <ChestPainType>4</ChestPainType>
    <RestingBloodPressure>138</RestingBloodPressure>
    <SerumCholestoral>183</SerumCholestoral>
    <FastingBloodSugar>0</FastingBloodSugar>
    <RestingElectrResults>0</RestingElectrResults>
    <MaxHeartRate>182</MaxHeartRate>
    <ExerciseInducedAngina>0</ExerciseInducedAngina>
    <Oldpeak>1.4</Oldpeak>
    <SlopeOfPeakExerciseSTSegment>1</SlopeOfPeakExerciseSTSegment>
    <NumberMajorVessels>0</NumberMajorVessels>
    <Thal>3</Thal>
  </example>
</hea1>
</td>
<td>
<hea3>
  <example>
    <class>2</class>
    <FBSugar>0</FBSugar>
    <REResults>0</REResults>
    <SlopePESTS>1</SlopePESTS>
    <S>1</S>
    <CPT>2</CPT>
    <MaxHR>141</MaxHR>
    <EIA>0</EIA>
    <OP>0.3</OP>
    <MajVesselsNo>0</MajVesselsNo>
    <Thal>7</Thal>
    <Years>57</Years>
    <RBPress>124</RBPress>
    <SChol>261</SChol>
  </example>
</hea3>
</td>
<td>
class,class
Sex,S
example,example
Thal,Thal
RestingBloodPressure,RBPress
SerumCholestoral,SChol
MaxHeartRate,MaxHR
RestingElectrResults,REResults
ChestPainType,CPT
FastingBloodSugar,FBSugar
SlopeOfPeakExerciseSTSegment,SlopePESTS
ExerciseInducedAngina,EIA
NumberMajorVessels,MajVesselsNo
Age,Years
Oldpeak,OP
</td>
</tr>
</table>

2.1. The XMapper Architecture

Two XMapper system has two modules:

- *constraints analysis* module, which is used to extract properties of data stored in XML sources, like data types, length, number of null values etc., and structural information, like number of children nodes, data types of children nodes etc.
- *learning module*, that is used to extract information about relationship between attributes used in both data sources.

The learning module uses inductive ML algorithm DataSqueezer [10] to generate six feature values from a vector of 22 features that describe an attribute. The DataSqueezer is a supervised algorithm that generates production rules. In case of the XMapper, it will use flat data stored within an XML document as its training data, and the user-specified attribute (tag) as a class label. To describe the DataSqueezer algorithm we denote the set of all training examples by S, the set of positive examples as S_P, and negative examples as S_N. Examples are described by a set of K attribute-value pairs [23, 24]: $e = \wedge_{j=1}^{K}[a_j \# v_j]$ where a_j denotes j-th attribute with value $v_j \in d_j$, # is a relation (=, <, ≈, ≤, etc.), where K is the number of attributes. In case of the DataSqueezer algorithm the relation is equality. An example, e, consists of set of selectors $s_j = [a_j = v_j]$. The DataSqueezer algorithm generates production rules in the form of:

IF ($s_1 \wedge \ldots \wedge s_m$) THEN class = $class_i$,
where $s_i = [a_j = v_j]$ is a single selector, and $m \leq K$

The DataSqueezer generates rules using two-phase process. First, common selectors for examples from S_p are found. Next, the rules are generated and validated against S_N using the strongest found selectors. Each generated rule has an assigned goodness value that is equal to the percentage of positive examples from the training data that it covers. Based on these goodness values, a goodness for each attribute and selector from the data is computed using a procedure described in [9]. For detailed description see [10]. The goodness values computed for attributes and selectors by the DataSqueezer algorithm are used by the XMapper system to derive six feature values for every attribute from both XML sources.

The diagram of the XMapper architecture is shown in Figure 1. Detailed description together with a pseudo-code of the XMapper system is given in the next section.

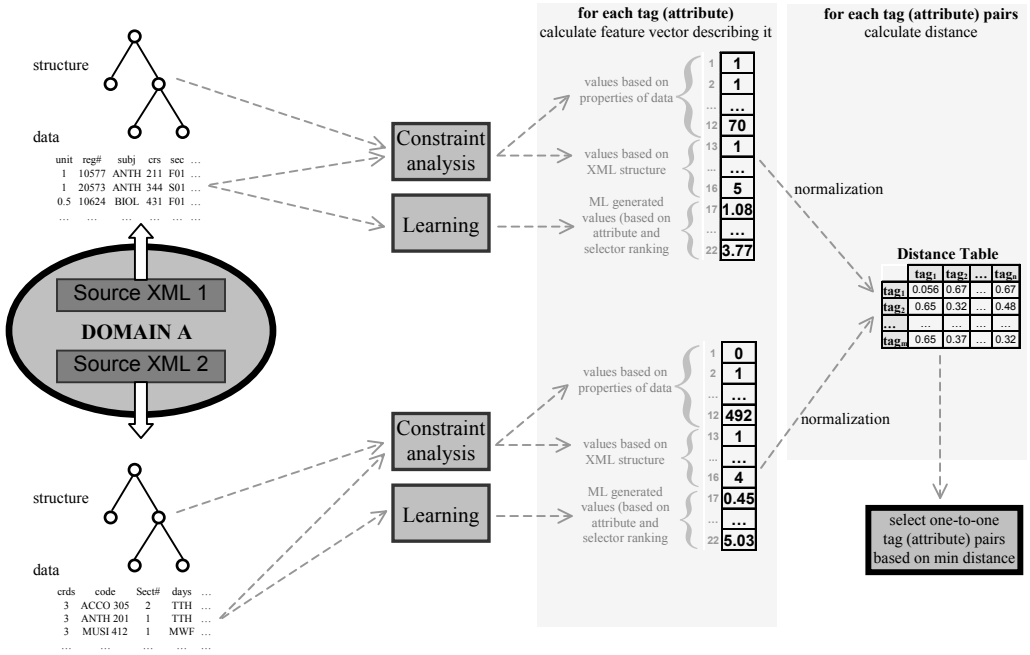

Figure 1 The architecture of the XMapper system

2.2. The XMapper Algorithm

The XMapper pseudocode:

Given: Two XML sources, two user-selected class attributes (tags), one per each source

Initialize: For both XML sources extract:

- data as flat data tables where rows are data examples and columns are attributes
- tree, in form of list of corresponding parent-children nodes

Step1.

1.1 For both XML sources calculate feature vectors describing their attributes
  for i^{th} XML document
    for j^{th} attribute
      calculate 22 dimensional feature vector $v_{i,j}$

1.2 Calculate distance table by calculating distance between all attribute pairs between the two XML sources
  for i^{th} attribute from XML document 1
    for j^{th} attribute from XML document 2
      calculate $dist_{i,j}$ = distance between $v_{1,i}$ and $v_{2,j}$

Step2.

Initialize: L empty list of attribute pairs

2.1 find max = maximum value of $dist_{i,j}$

2.2 find min = minimum value of $dist_{i,j}$

2.3 if min > 0.5*max then STOP

2.4 add (i, j) pair of attributes, which corresponds to the min, to L

2.5 set $dist_{k,j} = 1$
  where k = 1,2,…,#attributes of XML document 1
  set $dist_{i,k} = 1$
  where k = 1,2,…,#attributes of XML document 2

2.6 until all rows or columns of distance table contain only 1's go to 2.2

Output: L (list of 1-to-1 attribute pairs)

In step 1, 22 dimensional feature vectors that describe attributes from both XML sources are calculated. Also, the distances between feature vectors from different XML sources are computed. The values of the feature vectors that are calculated in step 1.1 are described in detail in Table 2. The last six values from these vectors are calculated using the learning module of the XMapper. Feature vectors are normalized into the [0; 1] range and multiplied by a weight. Weighted sum of all feature values is scaled to the [0; 1] range in order to compute distances between feature vectors in step 1.2.

Table 2. 22 dimensional feature vector describing an attribute

source	index	description	weight	source	index	description	weight
constrain analysis	1	type of values (1 if has some numerical only values, 0 otherwise)	0.025	XML structure	13	number of children attributes (nodes) all the way to leaf nodes	0.03
	2	type of values (1 if has some char only values, 0 otherwise)	0.025		14	most complex children attribute type of values	0.01
	3	type of values (1 if has some special char only values, 0 otherwise)	0.025		15	mean max length of all children attributes	0.01
	4	type of values (1 if has some num. and char values, 0 otherwise)	0.025		16	most common children attribute type	0.01
	5	type of values (1 if has some num. and special char values, 0 other.)	0.025			sum	0.06
	6	type of values (1 if has some char and special char values, 0 other.)	0.025	inductive learning	17	attribute goodness	0.12
	7	type of values (1 if has some num., char and spec char values, 0 other.)	0.025		28	min. goodness of attribute selectors	0.04
	8	type of values (1 if has some empty values, 0 other.)	0.025		19	max. goodness of attribute selectors	0.04
	9	max. length of values (0 empty, 1 single char, floor(2+max length/10))	0.04		20	mean goodness of all children attributes	0.12
	10	attribute type (0 discrete, 1 continuous)	0.04		21	mean min. goodness of all selectors of all children attributes	0.04
	11	number of empty values	0.03		22	mean max. goodness of all selectors of all children attributes	0.04
	12	number of distinct values	0.03			sum	0.4
		sum	0.34				

The distance between i^{th} and j^{th} attribute, which is calculated in step 1.2, is defined by the following formula:

$$dist_{i,j} = \sum_{k=1}^{22} (w_k F_{i,k} - w_k F_{j,k}) + 0.1 \cdot a + 0.1 \cdot b \quad (1)$$

where

$F_{i,k}$ is scaled value of k^{th} feature of i^{th} attribute

w_k is weight value of k^{th} feature

$$a = \begin{Bmatrix} 1 \text{ if } i^{th} \text{ tag name is a substring of } j^{th} \text{ tag name} \\ 0 \text{ otherwise} \end{Bmatrix}$$

$$b = \begin{Bmatrix} 1 \text{ if } i^{th} \text{ tag name is an abbreviation of } j^{th} \text{ tag name} \\ 0 \text{ otherwise} \end{Bmatrix}$$

During computation of the distance, the values of the 22 feature vectors together with corresponding weights are used. Distance value also incorporates mechanism of matching attribute names. The matching of attributes names uses two measures:

- matching the entire attribute names using formula for *a* in (1). XMapper checks if name of one attribute is identical or substring of the other attribute name.
- matching abbreviation of attribute names using formula for *b* in (1). XMapper checks if name of one attribute is an abbreviation of the other attribute name.

Distance values are normalized to the [0, 1] range to enable comparison of distances between different pairs of attributes.

In step 2, pairs of attributes belonging to different XML sources that have minimum distance are chosen. To decide if an attribute pair with the minimum distance should be output as a mapping, the XMapper uses in step 2.3 the stop threshold. The stop threshold assures that attributes that are not similar to any other attributes, which corresponds to high value of distance between them and other unmatched attributes, will not be not mapped.

Since the DataSqueezer algorithm that is used in the learning module of the XMapper system work only with discrete and numerical data, two data transformation are performed:

- numerical encoding. All textual attribute values are encoded into numerical values. Since the DataSqueezer is not distance sensitive, simple encoding scheme when a text string is encoded into an arbitrary value is performed
- discretization. All attributes with high number of encoded values are discretized to reduce the number of values.

For deciding if an attribute is continuous the following rule of thumb is used [5]:

$$n_i = \frac{N}{3c}$$

where

n_i number of discrete intervals for i^{th} attribute
N number of distinct values of i^{th} attribute
c number of classes (distinct values of class attribute)

The attribute is continuous, and thus need to be discretized, if $n_i > c$. In another words, an attribute is considered continuous if the number of discrete intervals calculated using the above rule of thumb is greater than the number of classes. Number of classes is equal to the number of distinct values of the user-specified attribute.

The unsupervised Equal Frequency discretization algorithm [6] was used to discretize an attribute. Unsupervised discretization was used because it is fast, and the user-specified attribute (class attribute) may not be meaningful (required for supervised

discretization). In the future, we plan to test a supervised discretization algorithm, like CAIM [20] or CADD [5] to improve the results generated by the learning module of the XMapper system.

One of the XMapper advantages is its flexibility of using any ML algorithm that generates ranked list of attributes and selectors, along with their goodness values. Example is the CLIP4 algorithm [7, 9], which can be used instead of the DataSqueezer algorithm. Also feature selection algorithms like the ReliefF [18] and the Relief [16, 17] can be used. They work by assigning a "relevance" weight to each attribute, which can be interpreted as a goodness measure. The disadvantage of the latter algorithms is that they do not provide selector ranking and thus the learning module of XMapper system would only generate first three out of the six feature values.

The XMapper system extracts more information about the XML source than is stored within a DTD document. For example, it extracts information about the number of null values and the type of attributes. This is one of the reasons why the XMapper generates more accurate mappings than systems that use only DTD information. In the next section we compare the XMapper system with the LDS system. The comparison shows that our system generates more accurate mappings, in spite of the fact that LSD uses several ML learners.

3. Experiments

The XMapper system was tested using several domains that consisted of several XML data sources. Number of XML documents per domain varied from 2 to 5. The benchmarking test of the XMapper system consist of two parts:
- tests that use artificially created XML based domains
- tests that use real-life XML based domains

3.1. Domains

The domains used in the benchmarking of the XMapper can be divided into two categories: artificial and real. The artificial domains were created by converting into the XML format datasets downloaded from the UC Irvine Machine Learning Repository [2]. Sample dataset that shows how the original, comma-separated format was converted into the XML is shown in Table 1. The real domains were downloaded from the LSD web site at http://www.cs.washington.edu/homes/anhai/lsd/lsd.html. The benchmarking results obtained for the real datasets were compared with results obtained by the LDS system. Summary information about all datasets is given in Table 3.

The reason for using both domains was that real domains include mostly textual data, while artificial domains include mostly numerical data.

Table 3 also includes information about the structural and semantic difference between data sources belonging to the same domains. Changes between data sources in the artificial domains include changing attribute names into the new names or different types of abbreviations, reordering and deletions of attributes. Creation of a set of artificial data sources belonging to the same domain consisted of random splitting the original dataset into n subset and conversion of these subsets into the XML format. Next, the attribute names, order and attribute deletions were performed as shown in Table 3.

Description of the domains follows:

1. Contraceptive method choice (*cmc*)
2. StatLog heart disease (*hea*) (originally from the StatLog project repository)
3. Iris plant (*iris*)
4. Mushrooms (*mush*)
5. PIMA Indian diabetes (*pid*)
6. SPECT heart imaging (*spect*)
7. Thyroid disease (*thy*)
8. Course listing (*course*)
9. Faculty listing (*faculty*)
10. Real estate (*realest*)

First seven are the artificial domains. The *course* domain includes course listing from five universities. The *faculty* domain includes faculty listings from computer science departments from five universities. The *realest* domain includes house sale listings from five real estate sources.

Table 3. Major properties of datasets considered in the experimentation

domain	# sources	source files (datasets)	# listings (# exam)	# tags (# attrib)	# non-leaf tags	matchable tags	depth	tag properties
cmc	3	cmc1	491	11	1	100 %	2	original names, order and XML structure
		cmc2	491	10	1	100 %	2	changed: names and 1 removed
		cmc3	491	11	1	100 %	2	changed: names and order in XML
hea	3	hea1	90	15	1	100 %	2	original names, order and XML structure
		hea2	90	13	1	100 %	2	changed: names and 2 removed
		hea3	90	15	1	100 %	2	changed: names and order in XML
iris	2	iris1	75	6	1	100 %	2	original names, order and XML structure
		iris2	75	6	1	100 %	2	changed: names and order in XML
mush	3	mush1	2806	24	1	100 %	2	original names, order and XML structure
		mush2	2806	21	1	100 %	2	changed: names and 3 removed
		mush3	2804	24	1	100 %	2	changed: names and order in XML
pid	3	pid1	256	10	1	100 %	2	original names, order and XML structure
		pid2	256	9	1	100 %	2	changed: names and 1 removed
		pid3	256	10	1	100 %	2	changed: names and order in XML
spect	2	spect1	133	24	1	100 %	2	original names, order and XML structure
		spect2	134	24	1	100 %	2	changed: names and order in XML
thy	3	thy1	3772	23	1	100 %	2	original names, order and XML structure
		thy2	3428	22	1	100 %	2	changed: names, and order in XML, 1 removed
course	5	5 universities	703÷3924	15÷19	3÷5	58÷88	3÷5	course listing from 5 US universities (different names, order and XML structure)
faculty	5	5 universities	33÷74	14	4	100 %	4	faculty listing from 5 US universities (same names, order and XML structure)
realest	5	5 agencies	501÷3001	31÷53	1÷18	21÷44	2÷3	real estate listing from 5 agencies (different names, order and XML structure)

3.2. Results

The XMapper was tested on 10 domains that included between 2 and 5 XML sources. The benchmarking incorporated matching all possible pairs of XML sources for domains. For example, for the domain consisting of 5 sources, 10 tests were performed. The results include mean number of correctly and incorrectly generated mappings, and the mean accuracy. For each source pair we checked the results versus manually created list of correct mappings. The results incorporate all correctly generated mappings, which include 1-to-1 mappings and unmatched, single attributes that had no matching attribute in the other source. The incorrect mappings include incorrect 1-to-1 mappings and unmatched attributes that had a mapping. The results do not count the user-specified attribute (class attribute), which was not processed by the XMapper system.

Two sets of test were performed using all 10 domains:

- using the XMapper system with learning and constraints analysis modules
- using the XMapper system without the learning module.

This setup is intended to show the advantages of incorporating the learning module into the XMapper system. The test results are summarized in Table 4. Direct comparison of mean accuracies for all domains for both setups is shown in Figure 2.

The XMapper system achieved high, over 85%, accuracies for 7 out of 10 domains for both of the test setups. For these 7 domains results between the XMapper system with and without the learning module are comparable. Thus, for these domains it would be computationally less expensive to use the system without the learning module. There are several reasons for these results: the attribute names and types of attribute values for the mapped attributes within these domains and between sources were similar, which makes the mapping easier for the system.

For the remaining 3 domains (i.e. *spect, thy and realest*) the results while using the XMapper with the learning module are much better. For these domains the system achieved on average 62% accuracy, while without the learning module it achieved only 31% of accuracy. These results show the need for incorporating the learning module within the XMapper system.

There are several reasons for low performance of the XMapper system without the learning module for the three domains. The *spect* domain created using the *spect* dataset [19] had attributes with completely different names between the two sources. Also, all attributes were binary and thus only the relationship between attributes could be used as indicator for mapping them correctly.

Table 4. From left: benchmarking results for the Xmapper system using the leraning module, and without using the learning module

domain	sources	# experiments (source pairs)	mean # correct	mean # incorrect	mean acc. %
cmc	3	3	10	0	**100.0**
hea	3	3	12.3	1.7	**88.1**
iris	2	1	5	0	**100.0**
mush	3	3	19.7	3.3	**85.5**
pid	3	3	7.7	1.3	**85.1**
spect	1	1	15	8	**65.2**
thy	1	1	12	10	**60.0**
mean for artificial domains					83.4
course	5	10	15.5	2.8	**85.2**
faculty	5	10	13	0	**100.0**
realest	5	10	28.6	18.7	**60**
mean for real-life domains					81.7
total mean					**82.6**

domain	sources	# experiments (source pairs)	mean # correct	mean # incorrect	mean acc. %
cmc	3	3	10	0	**100.0**
hea	3	3	13.3	0.7	**95.2**
iris	2	1	5	0	**100.0**
mush	3	3	20.3	2.7	**88.4**
pid	3	3	8.3	0.7	**92.6**
spect	1	1	5	18	**21.8**
thy	1	1	3	19	**13.6**
mean for artificial domains					73.1
course	5	10	15.6	2.3	**87.5**
faculty	5	10	100	0	**100.0**
realest	5	10	27.4	20	**57.2**
mean for real-life domains					81.6
total mean					**77.3**

Similarly, the *thy* domain created using the *thyroid* dataset [28, 29] includes attributes that have very different attribute names, and very similar attribute values between the two sources. 92 percent of the examples from that domain belong to the same class. In case of the *realest* domain, only between 21 and 44% of attributes could be mapped. Also, there were many mappings that were very hard to distinguish between being semantically correct, and only similar in terms of structure and content. That resulted in higher number of incorrect mappings.

Using learning module resulted in improving accuracy of mappings for these hard domains. The ML component of the learning module improves the accuracy of results by incorporating information about the relationship between the attributes. This additional information is shown to improve the mapping accuracy. The total average accuracy results show that using the learning module improves the results by over 5%. It is important to notice that using the learning module improves the results for both artificial and real domains.

Another important advantage of the XMapper is that it returns ordered, in terms of confidence, set of mappings. The attribute pairs with the highest confidence are first. The confidence level directly corresponds to the distance calculated between attribute pairs.

Using results from the benchmarking tests we performed a set of tests that validate the usefulness of the ordering generated by the XMapper system. The error distribution versus the returned position of the mapped attributes was computed for the four discrete intervals: 0-25%, 25-50%, 50-75%, and 75-100%. The 0-25% interval corresponds to the first 25% percent of the returned mappings. The results are shown in Figure 3.

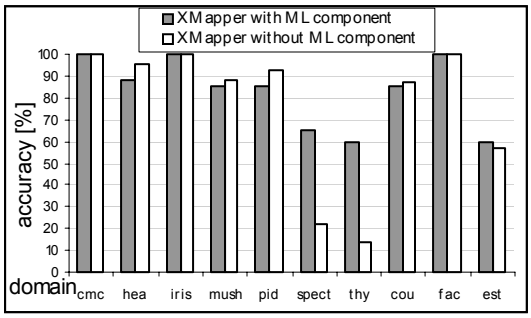

Figure 2. Comparison of results for the XMapper system with and without using the learning module

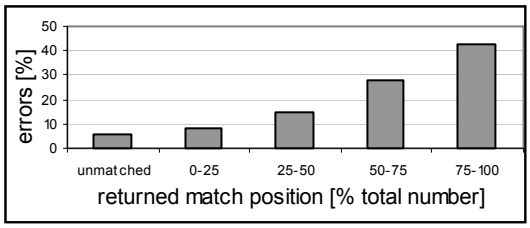

Figure 3. The error distribution for the ordered set of mappings returned by the XMapper system

The results show that the confidence measure used to order mappings generated by the XMapper provides valuable information to the user. The 71% of erroneous mappings are generated with the second half of the returned mappings, and over 40% for the last 25% of them. On the other hand, only 8% of errors are made within the first 25% of returned mappings. Also, since the XMapper made on average only 6% (Figure 3) of errors for attributes that were returned unmatched, the mechanism of using the stop threshold to generate unmatched attributes is also a robust solution.

The XMapper system was compared with the LSD system [12]. Both systems used ML to generate mappings. Only the real domains were used for the comparison since the LSD system was tested only on them. The comparison results are shown in Table 5. The main difference between the XMapper and LDS is that the LDS generated mappings into the mediated schema, which makes the task of finding mappings easier, while XMapper generated mappings directly between two sources. For *course* and *faculty* domains the XMapper generated mappings with higher accuracy. For the *realest* the XMapper results were worse than the LDS results. One of the reasons why the XMapper mapping were less accurate is that it could on average map only between 21 and 44% of attributes in this domain, since the matching were performed between the XML sources. The LDS on average could map between 84 and 100% of attributes since it maps attributes between a source and a mediated schema.

Table 5 Comparison of results between XMapper and LSD systems

	course	Faculty	Realest
LSD	76%,	92%	71%
XMapper	85%	100%	60%

The average accuracy comparison between the LSD and XMapper shows that XMapper generates mapping with higher (81.7%) accuracy than LSD (79.6%). The XMapper generates more accurate mappings despite solving more complex problem of direct mapping between sources, and using only source XML files, while LDS uses both DTD and source data information.

4. Future Work

Future work will include investigating the advantages of using DTD or XML schema information to reduce computational complexity of feature extraction by the constraint analysis module of the XMapper system. The XMapper currently extracts more information than is stored in a DTD document, and thus we will use the XML Schema, which is a relatively new and thus not yet widely used standard. Also, we are planning to enhance capabilities of the XMapper system by analysis and discovery of complex schema mapping using a data driven approach framework [32].

5. Summary and Conclusions

We introduced a new system, called XMapper, which generates semantic mapping of XML tags between two source documents. The XMapper uses both the structure and data information to generate the mappings.

The proposed system has several advantages. It can generate mappings in fully automated manner, without involving the user, except for input information that consists of a single pair of tags that correctly maps between the two sources. The XMapper system uses standalone XML only, and thus eliminates the need for creating DTD or Schema files that describe the XML sources. It generates mappings between all, including non-leaf, tags in contrast to other mapping systems, like the LDS system.

The XMapper system generates mappings with high degree of accuracy. Another advantage of the XMapper is that it returns ordered, in terms of confidence, mappings that can significantly help the user to discover incorrect mappings. It is also capable of returning both matched and unmatched tags.

In the nutshell, the XMapper system generates ordered set of semantic mappings between two XML source documents with high degree of accuracy. The developed system can help in building data integration system by automatically providing the system designer with easy to verify information, which can be used to merge the content of information coming from different sources.

References

[1] Bergamaschi, S., Castano, S., Vimeracati S.D.C.D. & Vincini, M., An Intelligent Approach to Information Integration, *Proceedings of the International Conference on Formal Ontology in Information Systems (FOIS-98)*, pp.253-267, Trento, Italy, 1998

[2] Blake, C.L. & Merz, C.J., UCI Repository of machine learning databases [http://www.ics.uci.edu/~mlearn/ MLRepository.html], Irvine, CA: University of California, Department of Information and Computer Science, 1998

[3] Bray, T., Paoli, J., and Maler E., Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation, http://www.w3.org/TR/ 2000/REC-xml-20001006, October 2000

[4] Castano, S. & Antonellis, V.D., A Schema Analysis and Reconciliation Tool Environment for Heterogeneous Databases, *Proceedings of the International Database Engineering and Applications Symposium (IDEAS-99)*, pp. 53-62, Montreal, Canada, 1999

[5] Ching, J.Y., Wong, A.K.C., & Chan, K.C.C., Class-Dependent Discretization for Inductive Learning from Continuous and Mixed Mode Data, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17:7, pp. 641-651, 1995

[6] Chiu, D., Wong, A., & Cheung, B., Information Discovery through Hierarchical Maximum Entropy Discretization and Synthesis, In: Piatesky-Shapiro, G., & Frowley, W.J., (Eds.) *Knowledge Discovery in Databases*, MIT Press, 1991

[7] Cios K. J. & Kurgan L., Hybrid Inductive Machine Learning: An Overview of CLIP Algorithms, In: Jain L.C. & Kacprzyk J. (Eds.) *New Learning Paradigms in Soft Computing*, Physica-Verlag (Springer), pp.276-322, 2001

[8] Cios K. J. & Kurgan L., Trends in Data Mining and Knowledge Discovery, In: Pal N.R., Jain, L.C. & Teoderesku, N. (Eds.), *Knowledge Discovery in Advanced Information Systems*, Physica-Verlag (Springer), accepted, 2002

[9] Cios K.J. & Kurgan L., Hybrid Inductive Machine Learning Algorithm that Generates Inequality Rules, *Information Sciences,* Special Issue on *Soft Computing Data Mining*, accepted, 2002

[10] Cios K. J. & Kurgan L., DataSqueezer Algorithm that Generates Small Number of Short Rules, submitted, 2002

[11] Clifton, C., Housman, E. & Rosenthal, A., Experience with a Combined Approach to Attribute-Matching Across Heterogeneous Databases, *Proceedings of the IFIP Working Conference on Data Semantics (DS-7),* Leysin, Switzerland. 1997

[12] Doan, A., Domingos, P. & Levy A., Reconciling Schemas of Disparate Data Sources: A Machine Learning Approach, *Proceedings of the SIGMOD Conference*, pp. 509-520, Santa Barbara, CA, 2001

[13] DTD Guide, Guide to the W3C XML Specification ("XMLspec") DTD, Version 2.1 http://www.w3.org/XML/ 1998/06/xmlspec-report-v21.htm, 2000

[14] Garcia-Molina, H., Papakinstantinou, Y., Quass, D., Rajaraman,, A., Sagiv, Y., Ullman, J. & Widom, J., The TSIMMIS Project: Integration of Heterogeneous Information Sources, *Journal of Intelligent Information Systes,* 8:2, pp.117-132, 1997

[15] ISO, ISO 8879:1986. *Information processing - Text and office systems - Standard Generalized Markup Language (SGML)*, 1986

[16] Kira, K. & Rendell, L.A., The Feature Selection Problem: Traditional Methods and a New Algorithm, *Proceedings of the 10th National Conference on Artificial Intelligence*, pp. 129-134, MIT Press, 1992

[17] Kira, K. & Rendell, L.A., A Practical Approach to Feature Selection, *Proceedings of the Ninth International Workshop on Machine Learning*, pp. 249-256, Aberdeen, Scotland, Morgan-Kaufmann, 1992

[18] Kononenko, I., Estimating Attributes: Analysis and Extensions of Relief, *Proceedings of the 1994 European Conference on Machine Learning*, pp. 171-182, Catania, Italy, 1994

[19] Kurgan, L.A., Cios, K.J., Tadeusiewicz, R., Ogiela, M. & Goodenday, L.S., Knowledge Discovery Approach to Automated Cardiac SPECT Diagnosis, *Artificial Intelligence in Medicine*, 23:2, pp 149-169, 2001

[20] Kurgan L. & Cios K.J., Discretization Algorithm that Uses Class-Attribute Interdependence Maximization, *Proceedings of the 2001 International Conference on Artificial Intelligence (IC-AI 2001)*, pp. 980-987, Las Vegas, Nevada, 2001

[21] Levy, A.Y., Rajaraman,, A. & Ordille, J., Querying Heterogeneous Information Sources Using Source Descriptions, *Proceedings of 22nd International Conference on Very Large Data Bases (VLDB)*, pp. 251-262 , Bombay, India, 1996

[22] Li, W. & Clifton, C., SEMINT: A Tool for Identifying Attribute Correspondence in Heterogeneous Databases Using Neural Networks, *Data and Knowledge Engineering,* 33, pp. 49-84, 2000

[23] Michalski, R. S., Discovering Classification Rules Using Variable-Valued Logic System VL1, *Proceedings of the 3rd International Joint Conference on Artificial Intelligence,* 162-172, 1973

[24] Michalski, R. S., A Theory and Methodology of Inductive Learning. In Michalski, R., Carbonell, J., & Mitchell, T.M. (Eds.), *Machine Learning,* Tioga Press, 1983

[25] Miller, R., Haas, L. & Hernandez, M., Schema Mapping as Query Discovery, *Proceedings of 26rd International Conference on Very Large Data Bases (VLDB)*, pp. 77-88, Cairo, Egypt, 2000

[26] Milo, T. & Zohar, S., Using Schema Matching to Simplify Heterogeneous Data Translation, *Proceedings of*

24rd International Conference on Very Large Data Bases (VLDB), pp.122-133, New York City, New York, 1998

[27] Ram, S. & Ramesh, H., Schema Integration: Past, Current and Future, In: Elmagarmid, A. et al. (Eds.) *Management of Heterogenous and Autonomous Database Systems*, pp.119-155, Morgan Kaufmann, 1999

[28] Schimann, W., Joost, M. & Werner, R., Synthesis and Performance Analysis of Multilayer Neural Network Architectures. Technical Report 16/1992, University of Koblenz, Institute of Physics, 1992.

[29] Schimann, W., Joost, M. & Werner, R., Optimization of the Backpropagation Algorithm for Training Multilayer Perceptrons, Technical Report, Institute of Physics, University of Koblenz, 1994

[30] Su, H., Kuno, H. & Rundensteiner, E.A., Automating the Transformation of XML Documents, *Proceedings of the 2001 Workshop on Web Information and Data Management (WIDM'01)*, pp.68-75, Atlanta, GA, 2001

[31] XML Schema, World Wide Web Consortium, XML Schema: Structures and Datatypes, http://www.w3.org/TR/xmlschema-1/ and http://www.w3.org/TR/xmlschema-2, May 2001

[32] Yan, L.L., Miller, R.J., Haas, L. & Fagin R., Data Driven Understanding and Refinement of Schema Mappings, *Proceedings of the SIGMOD Conference*, pp. 485-496, Santa Barbara, CA, 2001