

# Clustering XML Data by Structure and Values

## (Extended abstract)

Svetlozar Nestorov  
The University of Chicago  
evtimov@cs.uchicago.edu

### Abstract

A growing number of XML sources export large volumes of structurally-similar data with significantly different atomic values. Examples include product specifications, financial transactions, and medical records. We consider the problem of clustering such data using their structure such as attribute and element names and their nesting as well as attribute and element values. The result of the clustering can be used for data summarization and visualization and can also be relevant for storage, query optimization, and compression.

## 1 Introduction

The recent emergence and proliferation of industry-specific XML standards is bound to result in the accumulation of large amounts of XML data within public and private organizations. Example of such data sets include data collected by government agencies, financial transactions, medical records, sports scores, etc. A notable fact is that the European Union (EU) has mandated that by 2005 all financial reporting for EU-based companies must be in XBRL which is a financial XML standard.

Such XML data is often mostly non-textual. In other words, most of the attribute and atomic element values are numeric. This is in contrast with XML data found on the web which tends to contain mostly text.

Thus, XML is replacing a plethora of data formats used to report mostly numeric values. For such data the structures of individual objects are fairly similar but not identical. In contrast, in relational data all tuples from the same table have exactly the same structure, while in web XML data the structures vary widely. IN our case, small variations in the structure of the objects are important especially in the presence of a standard and collections of data from different entities.

In this paper, we present an initial approach to clustering similarly structured and mostly numeric XML data. In order to illustrate the problem consider the following example:

**Example 1** *Consider the following XML objects that describe three stock trades executed at some financial institution:*

```

<Trade id="A">           <Trade id="B">           <Trade id="C">
  <Price>102.5</Price>    <Price>99</Price>        <Price>102</Price>
  <Size>200</Size>       <Size>2000</Size>       <Size>250</Size>
  <Fee>190</Fee>        </Trade>                </Trade>
</Trade>

```

The first trade differs structurally from the other two because of the additional element *Fee*. Thus, based solely on the structure, *Trade B* and *Trade C* should be clustered together, separately from the first one. However, the values of the *Price* and *Size* elements of *Trade B* are rather different than the corresponding values of *Trade C*. In fact, the price and size of *Trade C* are quite similar to the price and size of *Trade A*. So, based solely on the values, *Trade A* and *Trade C* should be clustered together. Thus, in this example, the clues as to how to cluster the data given by the structure and the values point in different directions. Deciding which way to go is certainly application dependent but definitely not arbitrary.

Continuing with our example, suppose that the financial institution allows two types of accounts: one where a fee is paid for each trade and another with only a monthly fee. In this case, it makes sense to cluster the data using the price and size as well the existence or absence of a fee and its amount. Still, intuitively, *Trade A* and *Trade C* belong to one cluster and *Trade B* to another. However, if the size of *Trade C* were 1000 instead of 250, the existence of a fee in *Trade A* would have lead to clustering *Trade B* and *Trade C* together.

We conjecture that there is an inherent tradeoff between the difference in structure and values and that this tradeoff can be made explicit and parameterized. The exact choice of parameters, of course, remains application dependent.

The rest of the paper is organized as follows. In Section 2 we discuss background and related work. In Section 3 we outline the steps in the clustering process. The tradeoff between structure and value differences is examined in detail in Section 4. Finally, in Section 5 we discuss our next steps.

## 2 Background

There are several related methods that have been proposed for XML data. Structure inference techniques [5, 6, 4, 1] take into account only the structure of the data. Such technique are usually directed towards data with wide degree of varying structure. In contrast, we consider data where the structural differences, while meaningful, are relatively small. Also, we assume that the data conforms to a given schema, such as a DTD, while the goal of the above mentioned techniques is to find a schema. Several graph matching algorithms have been proposed for matching different schemas of XML objects. In our case, all objects conforms to the same schema, so there is no need find correspondence between object elements as it is already given. Recently, several compression algorithms for XML data have been proposed[3]. These algorithms also consider structure and values separately but do not (yet) consider the tradeoff between the two.

### 3 Clustering Process Outline

In this section we briefly describe the clustering process and enumerate the steps involved. We consider an iterative process, as most knowledge discovery tasks require [7], that uses the results obtained at one iteration to refine and calibrate the parameters and settings for the next iteration. The first step in this process is the preprocessing of the data. In this step we choose what parts of the data will be used by the clustering algorithm and perform data cleaning on these parts if necessary. The second step is the choice a framework that makes explicit the tradeoff between structure and value differences in the data and parameterize this tradeoff. This step is particular to XML; no such step is required for flat or relational data. The next step is to run a clustering algorithm that is appropriate for the specific application. Then, we close the feedback loop by examining the results and modify the preprocessing settings and/or the tradeoff parameters. The rest of this section elaborates on these steps.

1. **Preprocessing:** In this step, we need to determine what attributes and elements of the XML data will be used in the clustering. The choice is as difficult as with relational data for which we have the schema with the added complication that the XML schema is given as a DTD. Except for the required elements, we cannot tell from the DTD whether certain elements appear in the data, and if so how often. However, we can use some additional summarization or schema information such as data guides[6, 5] or typing[4]. As with relational data, we need to distinguish real-valued and categorical elements. With XML we will likely have some text elements as well.
2. **Structure and Value Tradeoff:** This step is the novel part in clustering XML data. The idea is to explicitly express the tradeoff between the differences in structure and value between XML data. The details are discussed in Section 4.
3. **Clustering Algorithm:** In this step, we choose the actual clustering algorithm. The choice is certainly application dependent and goal driven[2].
4. **Feedback:** The last step can serves as another starting point. Based on the actual results of the clustering algorithm we may decide to involve different set of attributes and elements, change the structure-value tradeoff framework, or refine its parameters.

### 4 Structure vs. Values Tradeoff

We consider two approaches to quantifying the tradeoff between structure and values. The first approach involves choosing a vector space and mapping each and every XML object to a point in this space. The advantage of this approach is that we can use directly any clustering algorithms that operates on vector spaces. In the second approach, we define a distance function between two XML objects. This approach does limit the choice of clustering algorithms since the our distance function may not have certain properties expected and exploited by some algorithms.

#### 4.1 Mapping to Vector Space

This approach involves two choices: a vector space and a function that maps an XML object to a point in this space. Note that during preprocessing we have already chosen the elements

and attributes that will be used the clustering. Thus, the natural choice of a vector space is one that has dimensions corresponding to the chosen attributes and elements. For the data in our running example, the vector space will have three dimensions, corresponding to price, size, and fee. Note that we choose not involve the `id` attribute in the clustering since it's sole purpose is to uniquely identify each trade.

The choice of a mapping function is a little trickier. Normalization issues aside<sup>1</sup>, the key question is what to do with objects that don't have all of the chosen attributes and elements. Note that we do not use the word *missing* for these elements. While in some cases the values of such elements may indeed be missing, in other cases these elements may be not applicable for the particular XML object. With relational data, it's often impossible to distinguish the two cases because a single value, such as `NULL`, is used to denote both cases. With XML however, the lack of an element altogether tends to indicate that the element is not appropriate, while a missing value is indicated by a `NULL`.

In the absence of any application specific information, a reasonable choice is to pick for every element  $l$  a value  $\omega_l$  so that objects with no  $l$  will be mapped to a point with  $\omega_l$  coordinate in the dimension corresponding to  $l$ . So, the mapping function is defined as follows:

**Definition 1** *Given elements  $l_1, \dots, l_n$  and values  $\omega_{l_1}, \dots, \omega_{l_n}$ , an XML object  $o$  that contains a subset of these elements is mapped to a point  $(v_1, \dots, v_n)$  where  $v_i$  is the value of  $l_i$  in  $o$  if  $o$  contains such element and  $\omega_{l_i}$  otherwise.*

In our running example, we only need to choose  $\omega_{fee}$  since all three trades have price and size elements. The three trades will be mapped to the following three-dimensional points:

	<i>price</i>	<i>size</i>	<i>fee</i>
Trade A	(102.5,	200,	190)
Trade B	(99,	2000,	$\omega_{fee}$ )
Trade C	(102,	250,	$\omega_{fee}$ )

Next, we consider the choice of  $\omega_{l_i}$ . Suppose the range of values of  $l_i$  in the data is  $[m_i, M_i]$ . Certainly,  $\omega_{l_i}$  should lie outside this interval, because it does not stand for a missing value but rather represents a difference in the structure of the XML object. Further insight into the choice of  $\omega_{l_i}$  can be gleaned from previous iterations. In general, we don't want  $\omega_{l_i}$  to be too far or too close to the value range as the following two examples illustrate.

**Example 2** *Consider the following XML objects that describe three stock trades executed at some financial institution:*

```

<Trade id="D">      <Trade id="E">      <Trade id="F">
  <Price>15</Price>  <Price>16</Price>  <Price>15.5</Price>
  <Size>1000</Size>  <Size>1200</Size>  <Size>1100</Size>
  <Fee>10</Fee>     <Fee>20</Fee>     </Trade>
</Trade>           </Trade>

```

*Suppose we pick  $\omega_{fee} = -1$ . Then the three trades will be mapped to the following points:*

---

<sup>1</sup>Normalization is usually a part of the clustering algorithm and ultimately application dependent.

	(price, size, fee)
Trade D	(15, 1000, 10)
Trade E	(16, 1200, 20)
Trade F	(15.5, 1100, -1)

Our intuition is that Trade D is more similar to Trade E than Trade F, but the Euclidean distances between the corresponding points support the opposite conclusion. The problem may be lessened with normalization but it still exists if we choose  $\omega_{fee}$  too close to the range of fee values. The main reason for this problem is that we trade the structural difference for too small of a value difference.

**Example 3** Consider the following XML objects that describe three stock trades executed at some financial institution:

```

<Trade id="G">      <Trade id="H">      <Trade id="I">
  <Price>15</Price>  <Price>15</Price>  <Price>20</Price>
  <Size>100</Size>   <Size>100</Size>   <Size>500</Size>
  <Fee>10</Fee>     </Trade>           </Trade>
</Trade>

```

Suppose we pick  $\omega_{fee} = -1000$ . Then the three trades will be mapped to the following points:

	(price, size, fee)
Trade G	(15, 100, 10)
Trade H	(15, 100, -1000)
Trade I	(20, 500, -1000)

Intuitively, Trade H is more similar to Trade G than Trade I but the Euclidean distances again lead to the opposite conclusion. The problem is that the structural difference dominates the value difference, i.e., we trade too much value for structure.

## 4.2 Defining Distance Function

The alternative to mapping XML to a vector space is to define a distance function directly on the XML data. The advantage of this approach is that we can explicitly account for the structural differences of the two particular XML objects for which we compute the distance. Recall that in the previous approach for all objects without a particular element we assigned the same value to the corresponding dimension. The problem with such uniform assignment is that relative distances between objects with the element and objects without the element can get skewed as Examples 2 and 3 show. One way to resolve this problem is to define a flexible mapping from an XML object to a vector space that *depends* on the another XML object. Thus, we can define the distance between two object as the distance between the mappings of each object with respect to the other. In this case, for every element  $l$  we pick a value  $\delta_l$  and make any object that contains  $l$  be  $\delta_l$  away from any object that does not contain  $l$  in the corresponding coordinate. Formally,

**Definition 2** Given elements  $l_1, \dots, l_n$  and values  $\delta_{l_1}, \dots, \delta_{l_n}$ , the distance between XML objects  $o$  and  $p$  that contain a subset of these elements is defined as the distance between  $(v_1, \dots, v_n)$

and  $(w_1, \dots, w_n)$  where:

- if both  $o$  and  $p$  contain  $l_i$  then  $v_i$  and  $w_i$  are the values of  $l_i$  in  $o$  and  $p$  respectively.
- if both  $o$  and  $p$  do not contain  $l_i$  then  $v_i = w_i = 0$ .
- if  $o$  contains  $l_i$  and  $p$  doesn't then  $v_i$  is the value of  $l_i$  in  $o$  and  $w_i = v_i + \delta_{l_i}$ .
- if  $p$  contains  $l_i$  and  $o$  doesn't then  $w_i$  is the value of  $l_i$  in  $p$  and  $v_i = w_i + \delta_{l_i}$ .

To illustrate this definition consider Trades **D** and **F** from Example 2 and suppose that  $\delta_{fee} = 500$ . Then the distance between them will be the distance between the following points:

	(price,	size,	fee)
Trade D	(15,	1000,	10)
Trade F	(15.5,	1100,	510)

## 5 Next Steps

Our next steps include running experiment over real-life and synthetic XML data with different clustering algorithms. The results of those experiments will guides us in refining the structure-value tradeoff framework. We also plan to consider the multiplicity of some elements which may result in several possible mapping of an XML object to a vector space. For example, a medical record may contain several blood pressure measurements and even though the measurements are ordered it is not clear how to compare such record to another record with a different number of such measurements.

## References

- [1] P. Buneman, S. Davidson, M. Fernandez, and D. Suciu. Adding structure to unstructured data. In *Proceedings of ICDT*, 1997.
- [2] D. Hand and H. Mannila and P. Smyth. *Principles of Data Mining*. The MIT Press, 2001.
- [3] H. Liefke and D. Suciu. XMill: an efficient compressor for XML data. In *Proceedings of the ACM SIGMOD International Conference*, 2000.
- [4] S. Nestorov, S. Abiteboul, and R. Motwani. Extracting schema from semistructured data. In *Proceedings of the ACM SIGMOD International Conference*, pages 295–306, Seattle, Washington, June 1998.
- [5] S. Nestorov, J. Ullman, J. Wiener, and S. Chawathe. Representative objects: Concise representations of semistructured, hierarchical data. In *Proceedings of ICDE*, pages 79–90, Birmingham, U.K., April 1997.
- [6] R. Goldman and J. Widom. DataGuides: Enabling query formulation and optimization in semistructured data. In *Proceedings of VLDB*, 1997.
- [7] U.M. Fayyad and G. Piatetsky-Shapiro and P. Smyth, editor. *Advances in Knowledge Discovery and Data Mining*. AAAI Press, 1996.