Schema Extraction and Integration of Heterogeneous XML Document Collections

Prudhvi Janga and Karen C. Davis

University of Cincinnati, Cincinnati, Ohio, USA
jangapi@mail.uc.edu, karen.davis@uc.edu

Abstract. The availability of vast amounts of heterogeneous XML web data motivates finding efficient methods to search, integrate, query, and present this data. The structure of XML documents is useful for achieving these tasks; however, not every XML document on the web includes a schema. We discuss challenges and accomplishments in the area of generation and integration of XML schemas. We propose and implement a framework for efficient schema extraction and integration from heterogeneous XML document collections collected from the web. Our approach introduces the Schema Extended Context Free Grammar (SECFG) to model XML schemas, including detection of attributes, data types, and element occurrences. Unlike other implementations, our approach supports the generation of XML schemas in any XML schema language, e.g., DTDs or XSD. We compare our approach with other proposed approaches and conclude that we offer the same or better functionality more efficiently and with greater flexibility.

Keywords: XML schema, schema integration, schema extraction, schema discovery.

1 Introduction

XML supports a wide range of web applications from intelligent web searching to web-based e-commerce. The availability of large amounts of heterogeneous and distributed web data from multiple XML data sources adds new challenges for the integration of XML data. In typical clustering or integration approaches, the structure or schema of the XML document should be known. However, most of the XML documents over the web have no accompanying schema. XML schema extraction helps not only in integration but also in efficient storage and querying of XML data [GGR+00, PLM+02, XP11]. Automatic schema extraction from XML documents also assists schema designers by letting them analyze schema patterns and find new ones [C02]. Extraction and integration of XML schemas facilitates XML data transformations to different formats such as relational data [MAC03]. Hence, there are several reasons to develop algorithms and methods for efficient automated generation and integration of XML schemas from a heterogeneous collection of XML documents.

Merged schemas should be concise but should also be able to completely represent the XML documents under consideration. As these two requirements are sometimes contradictory, finding an optimal tradeoff is a challenging task [C02, GGR+00]. XML

A. Cuzzocrea and S. Maabout (Eds.): MEDI 2013, LNCS 8216, pp. 176–187, 2013.

schema extraction also involves using XML schema languages such as DTD or XSD which incorporate the full expressive power of regular expressions, contributing to the complexities of efficient and accurate schema extraction.

In addition to the complexity, many of the research efforts that have proposed XML schema extraction techniques yield incomplete results. In our work we have proposed techniques for efficient extraction of XML schemas which not only generate schemas that are complete but are also capable of producing XML schemas in both popular schema languages, XSD and DTD, and could be extended easily to support other schema languages. Unlike other research efforts that only support schema generation on homogeneous collections, our techniques also support heterogeneous collections where the XML documents can have widely varying structures and content.

We develop the Schema Extended Context-Free Grammar (SECFG) that uses extended regular expressions to model an XML schema, thereby making the schema extraction process schema-language independent. We propose an algorithm that integrates individual schemas by merging SECFG grammars. The algorithms we propose for XML schema generation and integration address challenges such as data type, attribute, and property detection, as well as schema completeness and support for multiple schema languages. We propose a framework for our algorithms that outlines the process of extraction, clustering, integration, storage, and querying of XML web data.

The reminder of this paper is organized as follows. Section 2 describes related research. Section 3 discusses our approach for XML schema extraction from individual XML documents. Section 4 describes the schema unification/merge algorithm proposed for merging individual schemas. In Section 5 we present experimental results and conclude with a discussion of future work in Section 6.

2 Related Research

XML schema generation for a given XML document normally involves three main steps: extraction of XML elements, simplification/generalization to obtain the structure of the document, and transformation of the structure into an XML schema definition. Simplification or generalization of elements to generate good DTDs using regular expressions [GGR+00, AR05] only works with DTDs and does not address complete schema generation. Simplification or generalization of elements using tree construction has been proposed [JOK+02, MLN00], but there are no individual XML document schemas available except for the merged final schema after the process is complete. Several research efforts [C02, PV00, W95] use extended context-free grammars for modeling XML document structure. However, this approach does not produce a complete XML schema that represents the XML document collection and the extraction and integration of XML schemas are combined into a single stage. Min et al. [MAC03] propose a schema extraction system that supports the generation of a DTD and XSD from XML documents, but it does not guarantee the completeness of the generated schema. Xing et al. [XP11] focus on schema extraction from a large collection of XML documents, but it does not support complete XML schema generation, heterogeneous collections, and multiple schema languages.

XML schema integration merges the XML schemas into a single XML schema. If a single XML schema cannot be achieved, this step returns the merged schema along with other independent schemas that could not be merged. The XML schema integration step aims at generating a schema that is adequate to describe the collection of XML documents but not so generic that it could describe many other XML documents not under consideration [GGR+00]. There are many research efforts that have proposed techniques that work well with homogenous collections. Integration of XML schemas over a heterogeneous collection of XML documents is much more complicated than schema integration on a homogenous collection of XML documents.

The Xtract system proposed by Garofalakis et al. [GGR+00] uses factorization and the MDL principle to integrate candidate DTDs. The MDL principle [R78] states that the best theory or language (in this case, a DTD) that can be inferred from data (in this case, collection of XML documents) that follows some patterns is the one that minimizes the sum of the number of bits needed to describe the theory itself and the number of bits needed to encode data using that theory. Moh et al. [MLN00] use spanning graph construction to merge individual XML schemas that are represented by document trees; they do not detect attribute types and entity references or handle processing hyperlinks and multimedia data. Jung et al. [JOK+02] integrate XML schemas from homogenous XML document collections. Chidloviskii [C02] proposes schema induction from a set of range extended context free grammars. The author uses content and context similarity measures to merge non-terminal symbols present in the context-free grammars. Min et al. [MAC03] generate either a DTD or XSD schema which is the integrated schema of the XML documents collection.

Most of the previous techniques, summarized in Table 1, combine XML schema generation with schema integration. However, we perform XML schema generation as a separate step in our framework to avoid repeating it during both XML schema integration and clustering. Some of the research efforts discussed above do not yield a complete XML schema that represents the XML documents under consideration. The technique that we propose is capable of extracting XML schemas that are complete. Most of the research efforts are only capable of representing the extracted schema in either one of the schema languages (XSD or DTD). Since our approach for XML schema extraction is modular, the extracted schema information can be represented in any of the XML schema languages. All of the proposed integration techniques [JOK+02, C02, MAC03] suffer from the same problem. These techniques incur processing overhead in real time and cannot be used for large collections of XML documents. Many techniques work well with homogenous collections and do not address heterogeneous collections. The framework that we have proposed works well with large collections of both homogenous and heterogeneous data and produces an integrated view of the XML data by clustering similar XML document data together. The integration technique that we propose can produce one or more merged schemas from a given collection of XML documents based on the structure of documents present in a given collection. This helps to avoid over-fitting the given collection or failing to represent all the documents. Our framework also separates the extraction and integration of XML schemas into two different stages to achieve better real-time performance and guarantee reusability.

Table 1. Analysis of Research Efforts for Extraction and Integration of XML Schemas

	[GGR+00]	[JOK+02]	[MLN00]	[AR05]	[C02]	[XP11]	[MAC03]	Our Approach
schema type	DTD	DTD	DTD	DTD	XSD	DTD	DTD, XSD	DTD, XSD & others
complete schema	○	●	○	●	○	○	○	●
technique	Regular Expr. & Factoring	N-ary Trees	Document Trees & Spanning Graph	Regular Expr. & Factoring	Extended Context-Free Grammars	NFA to Lazy DFA	Restricted Content Model	Schema Extended Context-Free Grammars
accuracy or conciseness	MDL Theory	○	○	MDL Theory	○	MDL Theory	○	MDL Theory
heterogeneous collections	●	○	○	●	●	○	●	●
tag/attribute similarity	○	○	○	○	○	○	○	●
similarity measure	○	○	○	○	○	Edit Distance	○	Edit Distance
reusability of schemas	○	○	○	○	○	○	○	●

yes=●, no=○

3 XML Schema Generation

There are three main steps in our XML schema generation process. XML schema generation starts by ordering XML documents by search rank so that the most relevant schemas are extracted first. However, this is an optional step if XML schema generation is being carried out offline independently of a user search. Once that step is complete, for each XML document, the XML schema is obtained. The three main steps for XML schema generation are (1) structured example creation, (2) generation of an SECFG grammar, and (3) generalizing the grammar. Each of these steps is explained below.

3.1 Generation of Structured Examples

We represent XML documents as structured examples of an unknown SECFG to aid in the process of XML schema inference. Structured examples are derivation trees where all non-terminal labels are removed [C02]. We define a recursive algorithm that generates a structured example for a given element (*E*) present in an XML document. This algorithm initializes a temporary set of nodes *TN* to store the generated nodes of the derivation tree. The element *E* that has been supplied to the algorithm is parsed and all the sub-elements of *E* are extracted into *S*. If there are no sub-elements, the element is considered to be a simple element and the node for the element is generated using start and end tags of the element. However, if *E* is a complex element with sub-elements then the nodes for each of the sub-elements are generated recursively until the deepest level is reached. Once all the nodes for sub-elements inside the complex element have been generated, the algorithm then generates the node for the complex element itself and links it to the nodes of the sub-elements present in the

temporary set, *TN*. Thus, for a given XML document, *GenerateStructuredExample(E)* creates a derivation tree that is a combination of unknown nodes (that become non-terminal symbols in the grammar) as well as element nodes called the structured example of the document.

3.2 Schema Extended Context-Free Grammars

We model an XML schema by creating a new grammar called the Schema Extended Context-Free Grammar (SECFG), an extension of ECFG [C02]. We associate features of an XML schema with components of an SECFG as shown in Table 2. The SECFG grammar addresses features such as attribute detection, order of child elements, number of child elements and detection of default, fixed, and substitution group values for elements and attributes, along with all other features already addressed using ECFG modeling.

Regular expressions used to represent SECFG are called extended regular expressions because they support both the minimum and maximum number of occurrences as well as several other attributes of a given element with a properties tag. The properties inside a properties tag are defined as a semicolon-delimited list of strings where each string is denoted by $P=a$, where P represents the name of the property (attribute name) and a represents the value of the property. A properties tag that accompanies a terminal symbol summarizes features such as order, number of child elements, data type, default value, and fixed values.

Formally, a Schema Extended Context Free Grammar is defined by a 5-tuple $G = (T, N, P, \delta, start)$ where T, N, and P are disjoint sets of terminals, non-terminals, and properties, respectively. Each property in P is defined over an empty set or an enumeration or a range of values it can accept. When a property is defined over an empty set it can accept any values. The symbol *start* is an initial non-terminal and δ is a finite set of production rules of the form $A \rightarrow \alpha$ for $A \in N$, where α is an extended regular expression over terms, where each term is a terminal-properties-nonterminal-terminal sequence such as *tpBt'* where *t, t'* are a pair of opening and closing tags respectively, $t, t' \in T$, $B \in N$ and $p \in P$. The expression *tpBt'* can be abbreviated as *tp:B* without loss of information.

Table 2. Correspondence between XML Schema Language Features and SECFG Components

XML Schema Language	SECFG
Element name (tag)	Terminal
Element definition	Production
Element attributes (includes data types and occurrences)	Properties tag
Named complex type	Non-terminal
Abstract complex type	Non-terminal
Complex type definition	One or more productions
Sequence element group	Sequential pattern in a production
Choice element group	Disjunction pattern in a production

We induce an SECFG from a structured example by traversing the tree in a depth-first approach. Every time we encounter an unknown node in the structured example tree, we add a production to the SECFG being generated and traverse the unknown node under consideration to complete the right side of the production. Since the grammar represents the complete document structure, there are numerous productions that are structurally identical. To achieve conciseness, generalization removes duplicate productions and merges productions with similar content and context [C02].

3.3 Generalization of SECFG

Algorithm : ***GeneralizeSECFG***(G_i)

Input: grammar passed by reference: G_i

1: BEGIN
2: In G_i merge the nonterminal symbols having similar names and structure
3: Determine data types for terminals present in G_i
4: Determine and merge attributes for terminals present in G_i
5: Generalize content sets present in nonterminal productions of G_i into regular expressions.
6: Determine ranges(min,max occurence) of the generalized content sets in G_i
7: Generate extended regular expressions for content sets in G_i
8: END

Fig. 1. Generalization of an SECFG

This algorithm takes an SECFG G_i that has not been generalized. A generalized grammar is one in which data types, order, and other properties are defined and the nonterminal symbols with the same name are merged to yield a concise grammar that describes the structure of the XML document. When translated into a grammar, all the non-terminal and terminal tags are unique at this stage. Hence, there is no need for using tag similarity in generalizing a given SECFG. However, in the integration stage where multiple schemas are merged, we cannot guarantee uniqueness of elements and sub-elements among different schemas, so we measure tag similarity. The algorithm shown in Fig. 1 performs the generalization of an SECFG. Once the generalization is complete, we use an SECFG-to-schema mapping algorithm that associates the features of an XML schema with an SECFG as shown in Table 2 to derive an XML schema definition in any XML schema language such as DTD or XSD.

4 XML Schema Integration

XML schema integration involves merging multiple individual schemas into a final schema. We propose the algorithm shown in Fig. 2 for XML schema integration. We merge the grammars to produce an integrated SECFG. Any grammars that could not be merged in this stage are marked as independent grammars. Once the merging process is complete, we transform the integrated SECFG into an XML schema.

Algorithm : *XML Schema Integration*

Input: Collection of schema definitions: S
Output: Merged XML schema as XSD or DTD

```
1: BEGIN
2: Order the collection of XML schemas S by rank
3: Initialize an empty grammar G which holds the integrated schema grammar
4: for each schema S_i in S do
5:     Derive a schema extended context-free grammar G_i for S_i
6:     MergeGrammar(G_i, G) where G is the base grammer
7:     Generate XML schema definition S_def from G
8:     Transform S_def to desired schema language XSD or DTD
9: end for
10: END
```

Fig. 2. XML Schema Integration

Algorithm : *MergeGrammar*(G_i, G)

Input: grammars passed by reference: G_i and G

```
1: BEGIN
2: Initialize an empty grammar G_temp
3: Calculate similarity measure Sim_{G_i,G} ▷ Edit distance is used to calculate similarity
4: if Sim_{G_i,G} ≥ Sim_threshold then      ▷ Sim_threshold is max. edit distance allowed
5:     Add productions of G to G_temp
6:     Add productions of G_i to G_temp
7:     GeneralizeSECFG(G_temp)
8:     G ← G_temp
9: else
10:    G_i.IsIndependent=true
11: end if
12: END
```

Fig. 3. Merge Grammar G_i into G

The algorithm shown in Fig. 3 merges a given grammar G_i into a base grammar G (the grammar that holds the integrated structure) and also generalizes the merged grammar. It starts by initializing a grammar G_{temp} that stores all the productions of G_i and base grammar, G. The grammar G_{temp} is generalized to give a compact structure stored as G. Since we consider a set of heterogeneous XML documents, the grammar G_i produced by the XML document X_i might be structurally different from the base grammar, G. We calculate the edit distance between the base grammar G and the grammar to be merged, G_i. If the edit distance is less than or equal to the maximum edit distance parameter, we merge the two grammars. We define edit distance as the minimum cost of the edit scripts such as insert a non-terminal, delete a non-terminal, or replace a non-terminal that transform the individual SECFG grammar to conform to the base grammar. As the edit distance might differ based on the size of the document, we use a normalized edit distance measure to compare to the maximum edit distance. Once the similarity between the base grammar and the grammar G_i is

calculated, we check whether it is above the threshold defined by the end user. If it is, then we merge the two grammars.

5 Experimental Results

In this section we discuss results of experiments carried out to test the validity and efficiency of our approach.

5.1 Comparison of an Actual versus Generated XML Schema

We generate XML schemas from a number of different XML datasets that have been automatically created using Altova XMLspy. We use synthetic as well as real world schemas in our experiments. Fig. 4 shows an example of an actual XML schema that represents a collection of XML documents and the one that is generated from our system by extracting and integrating schemas from the collection of XML documents. We can observe that both the schemas are very close to each other, differing mainly in the number of maximum occurrences of the element <*Title*> (10 in the original and 6 in ours.) The exact number of minimum and maximum occurrences cannot be guaranteed by any approach if the collection of XML documents does not fully represent the schema under consideration. Results from the experiments are discussed in the next section.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<!-- definition of simple elements -->
<xs:element name="Instructor" type="xs:string"/>
<xs:element name="Capacity" type="xs:positiveInteger"/>
<xs:element name="Room" type="xs:string"/>
<xs:element name="Title" type="xs:string"/>

<!-- definition of attributes -->
<xs:attribute name="courseid" type="xs:string"/>

<xs:element name="Course">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Title" minOccurs="0" maxOccurs="10"/>
      <xs:element ref="Instructor" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Capacity" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Room" minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
    <xs:attribute ref="courseid" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="Courses">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Course" minOccurs="1" maxOccurs="1000"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

Fig. 4(a). Example of an Actual XSD

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <#comment/>
    <xs:element name="Title" type="xs:TextOnly"/>
    <xs:element name="Instructor" type="xs:TextOnly"/>
    <xs:element name="Capacity" type="xs:TextOnly"/>
    <xs:element name="Room" type="xs:TextOnly"/>
    <#comment/>
    <xs:attribute name="courseid" type="xs:Text"/>
    <#comment/>
    <xs:element name="Course" type="xs:ElementsOnly">
        <xs:complexType>
            <xs:Sequence>
                <xs:element ref="Title" maxOccurs="6" minOccurs="0"/>
                <xs:element ref="Instructor" maxOccurs="1" minOccurs="0"/>
                <xs:element ref="Capacity" maxOccurs="1" minOccurs="0"/>
                <xs:element ref="Room" maxOccurs="1" minOccurs="0"/>
            <xs:attribute ref="courseid" use="required" default="String"/>
    <xs:element name="Courses" type="xs:ElementsOnly">
        <xs:complexType>
            <xs:Sequence>
                <xs:element ref="Course" maxOccurs="1000" minOccurs="1"/>
```

Fig.4(b). Our Generated XSD

5.2 Datasets and Quality of Inferred Schemas

To validate our approach and also show the advantages of it, we compare the schemas generated using our approach with XTRACT [GGR+00], DDbE [LA01], DTDXtract [AR05], and the restricted element content model [MAC03]. Although there are some other systems that also discuss XML schema generation, we do not consider them for comparison because they either do not discuss their algorithms in detail or they do not present experimental details. We have implemented our approach using Microsoft Visual Basic and the .Net framework. We use the same real-life DTDs in the experiments with XTRACT [GGR+00]. For each DTD for a single element, we generate an XML file containing 1000 instantiations of the element using Altova XMLspy. The regular expressions that represent the original DTDs used to generate the datasets are shown in the first column of Table 3. We refer to the regular expression representing a DTD as the DTD itself from here on. The DTDs from other approaches such as XTRACT, DDbE, and DTDXtract were obtained from Min et al. [MAC03]. The results from our approach are shown in the rightmost column in Table 3. Our approach generates all the schemas that match the original DTDs, demonstrating that our approach is accurate and valid. One noteworthy difference is that our approach produces schemas (can be represented in XSD or DTD or any other schema language) that match the original DTDs while other approaches just produce DTDs.

Table 3. Comparison of Actual Versus Generated DTDs

Original DTD	XTRACT	DTDXtract	DDbE ver2	Restricted Element Content Model	Our Approach
a|b|c|d|e	*a|b|c|d|e*	*a|b|c|d|e*	*a|b|c|d|e*	*a|b|c|d|e*	*a|b|c|d|e*
*(a|b|c|d|e)**	*(a|b|c|d|e)**	*(a|b|c|d|e)**	*((a|(e|b|c|a|d|(d+c))|(e|a|(e|c|d|(e+b+))|c|b|(e+b+)|d|(d+c))|d|e|b|c|(e+b+)|(dcb)|(d+c))*)*	*(a|b|c|d|e)**	*(a|b|c|d|e)**
*ab*c**	*ab*c**	*ab*c**	*(a(b|c)+)*	*ab*c**	*ab*c**
*a*b?c?d?*	*a*b?c?d?*	*a*b?c?d?*	*(((a|b)|c|d|a|((b|c)|d|b|c)|b)+)*	*a*b?c?d?*	*a*b?c?d?*
*(a(bc)+d)**	*(a(bc)*d)**	*(a(bc)+d)**	---	*(a(bc)+d)**	*(a(bc)+d)**
*(ab?c*d?)**	---	*(ab?c*d?)**	*((((ac+ac+d)|(a+b)|a)| . . . |(c+dab))*)*	*(ab?c*d?)**	*(ab?c*d?)**

5.3 Normalized Time Comparison of XML Schema Extraction Techniques

We compare the time taken to extract XML schemas from datasets among different systems such as XTRACT, DDbE, and our system. We exclude DTDXtract from the normalized time comparisons because this system uses the same algorithm used by XTRACT apart from an extension to support Kleene plus (+) expressions. Fig. 5 shows a normalized time comparison of different systems with respect to the different DTDs. We consider a normalized time comparison because the time taken to extract various DTDs by different systems such as XTRACT have been implemented using a computer configuration which is different from the configuration that we have used to

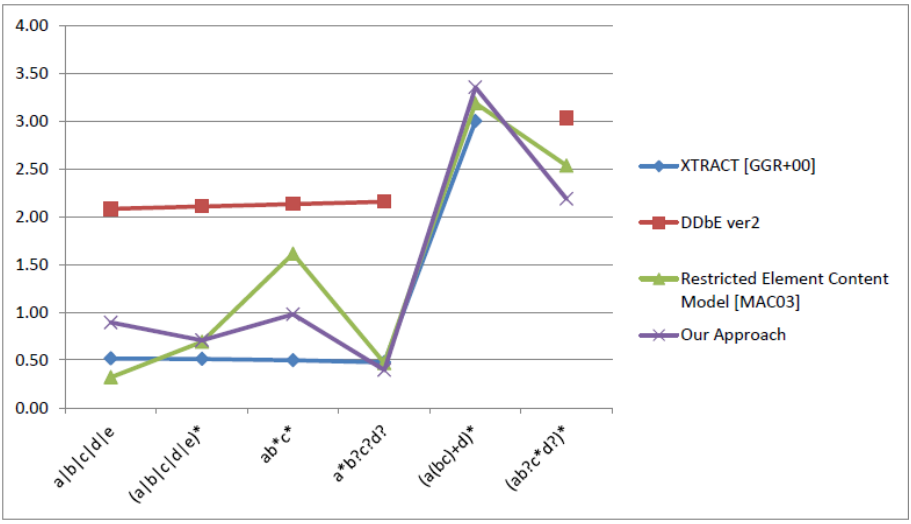

Fig. 5. Normalized Time Trends for DTD Generation

develop our system [MAC03]. Fig. 5 shows the standard scores (how many standard deviations an observation is above or below the mean). The time taken for XSD as well as DTD generation for different DTD/XSD data sets in our system is significantly less when compared to other systems such as XTRACT and DDbe ver2. Our system is not only comparable to the XTRACT system but also handles one additional case. Our system performs as well as or better than the element content model. It should also be noted that the time taken by our system is comparable or better than other systems even though it produces both DTD and XSD schemas. This result validates the efficiency of the approach we have taken as well as the system we have implemented.

6 Conclusions and Future Work

We introduce the problem of extraction and integration of XML schemas from heterogeneous collections of XML documents. We propose a grammar to represent the structure of the XML document (SECFG) and implement algorithms to extract and integrate XML schemas. We describe experimental studies that evaluate the effectiveness of our schema extraction and integration algorithms. We perform experiments using synthetic as well as real-life datasets and compare results from our approach against other systems that have already been proposed.

We discuss challenges involved in the extraction and integration of XML schemas and techniques to overcome those challenges. However, we have not addressed challenges involved in the location or clustering of XML web data. We use normalized edit distance to calculate the similarity between an individual schema and the base schema (to be the final schema) during schema integration. Future work could be done on enhancing our schema integration algorithm by incorporating some hybrid similarity measures (schema mapping techniques) that have been proposed for XML document clustering purposes. We are also working on using SECFG model for XML to relational schema mapping.

We expect to obtain additional experimental results as evidence of the validity and scalability of our approach; we plan to release our datasets to support reproducibility of our results as well as further research when this is complete.

References

[AR05] Leonov, A.V., Khusnutdinov, R.R.: Study and Development of the DTD Generation System for XML Documents. Programming and Computer Software (PCS) 31(4), 197–210 (2005)

[C02] Chidlovskii, B.: Schema extraction from XML collections. In: Proceedings of the 2nd ACM/IEEE-CS Joint Conference on Digital Libraries, Portland, Oregon, USA, June 14-18, pp. 291–292 (2002)

[GGR+00] Garofalakis, M.N., Gionis, A., Rastogi, R., Seshadri, S., Shim, K.: XTRACT: A system for extracting document type descriptors from XML documents. In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, Dallas, Texas, USA, May 16-18, pp. 165–176 (2000)

[JOK+02] Jung, J.-S., Oh, D.-I., Kong, Y.-H., Ahn, J.-K.: Extracting Information from XML Documents by Reverse Generating a DTD. In: Proceedings of the 1st EurAsian Conference on Information and Communication Technology (EurAsia ICT), Shiraz, Iran, October 29-31, pp. 314–321 (2002)
[LA01] Berman, L., Diaz, A.: Data Descriptors by Example (DDbE), IBM alphaworks (2001), http://www.alphaworks.ibm.com/tech/DDbE
[MAC03] Min, J.-K., Ahn, J.-Y., Chung, C.-W.: Efficient Extraction of Schemas for XML Documents. Information Processing Letters 85(1), 7–12 (2003)
[MLN00] Moh, C.-H., Lim, E.-P., Ng, W.K.: DTD-Miner: a tool for mining DTD from XML documents. In: Proceedings of the Second International Workshop on Advance Issues of E-Commerce and Web-Based Information Systems (WECWIS 2000), Milpitas, California, USA, June 8-9, pp. 144–151 (2000)
[PLM+02] Passi, K., Lane, L., Madria, S.K., Sakamuri, B.C., Mohania, M., Bhowmick, S.S.: A model for XML Schema Integration. In: Bauknecht, K., Tjoa, A.M., Quirchmayr, G. (eds.) EC-Web 2002. LNCS, vol. 2455, pp. 193–202. Springer, Heidelberg (2002)
[PV00] Papakonstantinou, Y., Vianu, V.: DTD Inference for Views of XML Data. In: Proceedings of the 19th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS), Dallas, Texas, USA, May 15-17, pp. 35–46 (2000)
[R78] Rissanen, J.: Modeling by shortest data description. Automatica 14(5), 465–471 (1978)
[W95] Wood, D.: Standard Generalized Markup Language: Mathematical and Philosophical Issues. In: van Leeuwen, J. (ed.) Computer Science Today. LNCS, vol. 1000, pp. 344–365. Springer, Heidelberg (1995)
[XP11] Xing, G., Parthepan, V.: Efficient Schema Extraction from a Large Collection of XML Documents. In: Proceedings of the 49th Annual Southeast Regional Conference, Kennesaw, GA, USA, March 24-26, pp. 92–96 (2011)