View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by TopSCHOLAR

Western Kentucky University
TopSCHOLAR®

Masters Theses & Specialist Projects

Graduate School

5-1-2011

Efficient Schema Extraction from a Collection of XML Documents

Vijayeandra Parthepan
Western Kentucky University, vijayeandra@gmail.com

Follow this and additional works at: http://digitalcommons.wku.edu/theses

Part of the Databases and Information Systems Commons, and the Programming Languages and Compilers Commons

Recommended Citation
Parthepan, Vijayeandra, "Efficient Schema Extraction from a Collection of XML Documents" (2011). *Masters Theses & Specialist Projects.* Paper 1061.
http://digitalcommons.wku.edu/theses/1061

This Thesis is brought to you for free and open access by TopSCHOLAR®. It has been accepted for inclusion in Masters Theses & Specialist Projects by an authorized administrator of TopSCHOLAR®. For more information, please contact connie.foster@wku.edu.

EFFICIENT SCHEMA EXTRACTION FROM A COLLECTION OF XML DOCUMENTS

A Thesis
Presented to
The Faculty of the Department of Mathematics and Computer Science
Western Kentucky University
Bowling Green, Kentucky

In Partial Fulfillment
Of the Requirements for the Degree
Master of Science

By
Vijayeandra Parthepan

May 2011

EFFICIENT SCHEMA EXTRACTION FROM A COLLECTION OF XML DOCUMENTS

Date Recommended May 28, 2011

[illegible]

Dr. Guangming Xing, Director of Thesis

[illegible]

Dr. Qi Li

[illegible]

Dr. Zhonghang Xia

Richard G. Bowker May 16, 2011

Dean, Graduate Studies and Research Date

ACKNOWLEDGEMENTS

It was a great pleasure working under my graduate advisor, Dr. Guangming Xing, who provided everything I needed to succeed. His inspiration and guidance made this Master of Science degree so rewarding. He always promoted my work and encouraged me to take the risks in my work and to express my ideas without any restrictions. I feel very fortunate to have been his student. I would like to thank Dr. Xing for the immense trust and patience he has over me. He constantly supported and directed me in each and every step. Without him this thesis would not have been a possibility.

I would like to thank Dr. Qi Li and Dr. Zhonghang Xia for their valuable time and suggestions that helped me improve this Thesis. I would like to thank all my friends at Western Kentucky University and my family for the never ending support. I would like to specially thank my father Mr. Parthepan Krishnaswamy and my mother Mrs. Geetharani Narayanaswamy for their everlasting love and encouragement for me to succeed.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

EFFICIENT SCHEMA EXTRACTION FROM A COLLECTION OF XML DOCUMENTS

Vijayeandra Parthepan May 2011 69 Pages

Directed by: Dr. Guangming Xing

Department of Mathematics and Computer Science Western Kentucky University

The eXtensible Markup Language (XML) has become the standard format for data exchange on the Internet, providing interoperability between different business applications. Such wide use results in large volumes of heterogeneous XML data, i.e., XML documents conforming to different schemas. Although schemas are important in many business applications, they are often missing in XML documents. In this thesis, we present a suite of algorithms that are effective in extracting schema information from a large collection of XML documents. We propose using the cost of NFA simulation to compute the Minimum Length Description to rank the inferred schema. We also studied using frequencies of the sample inputs to improve the precision of the schema extraction. Furthermore, we propose an evaluation framework to quantify the quality of the extracted schema. Experimental studies are conducted on various data sets to demonstrate the efficiency and efficacy of our approach.

CHAPTER 1: INTRODUCTION OF XML AND SCHEMA

1.1 Background

The eXtensible Markup Language (XML) [3] has become the standard format for data exchange on the Internet, providing interoperability between different business applications. Such wide use results in large volumes of heterogeneous XML data, i.e., XML documents conforming to different schemas. XML documents are naturally tree structured, and may contain atomic and complex structures. XML document can be viewed as an ordered labelled tree whose nodes are labelled with the name of the tags, and the children of each node are ordered. XML documents are also semi-structured as they incorporate both structural information and content [1]. Dealing with structure information is important to XML data storage and management [9], and the presence of a schema can significantly simplify the processing of XML documents. For example, elements from different documents with similar structures can be stored together, resulting in reduced storage and faster query processing. However, processing XML documents based on the schemas may not be feasible in practice, since most XML documents are schema-less.

`<catalogue>`	catalogue $\rightarrow$ book+
`  <book>`	book $\rightarrow$ title author+ year price
`    <title>Harry Potter</title>`	author $\rightarrow$ #PCDATA
`    <author>J K. Rowling</author>`	year $\rightarrow$ #PCDATA
`    <year>1998</year>`	price $\rightarrow$ #PCDATA
`    <price>16.99</price>`	
`  </book>`	
`  <book>`	
`    <title>XML</title>`	
`    <author>Author A</author>`	
`    <author>Author B</author>`	
`    <author>Author C</author>`	
`    <author>Author D</author>`	
`    <year>2011</year>`	
`    <price>10</price>`	
`  </book>`	
`</catalogue>`	

Figure 1.1: Catalog data and corresponding DTD structure.

An XML document is well-formed if it follows basic rules for XML such as,

1. There is exactly one element (root element) that contains all other elements.
2. Elements can be nested but may not overlap.

A well-formed document is valid if it contains a reference to a proper schema and the document conforms to the constraints of that schema. Validation requires an XML instance to contain all or some of the specified elements and attributes in the schema. An XML document is said to be valid against a schema if the elements and attributes in the XML document satisfy the constraints represented in the schema [18]. Based on the schema, an XML document can be transformed into other formats without content information. An XML document may include tags that describe the names of elements and contents such as text data and other forms of data.

XML schema is used to define the format and contents of a given XML instance [17], and it provides a mechanism to build relationships between schemas and elements. XML schema is a rigorous representation of an XML based language in terms of constraints on elements and attributes. The Document Type Definition (DTD) is the mostly widely used schema to specify the structure of XML documents. A sample XML document structure and the corresponding DTD can be illustrated by the following two figures here.

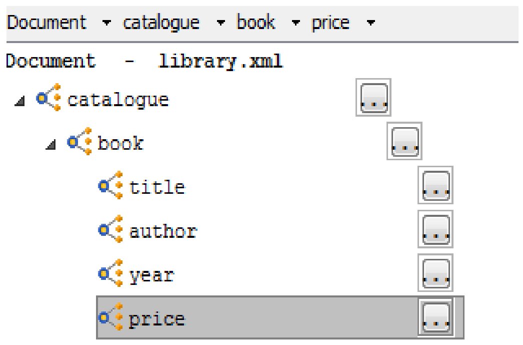


Figure 1.2: DTD document structure.

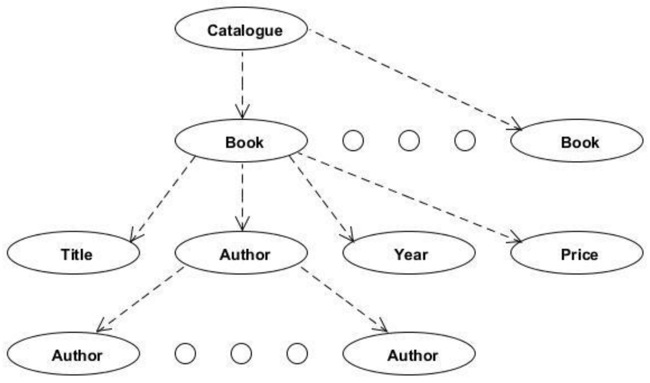


Figure 1.3: XML tree representation of a sample DTD.

1.2 XML Schema

1. XML documents are naturally tree structured, and incorporate both structural and content information [1] and the presence of a schema can significantly simplify the processing of XML documents.

2. The schema information can be used to group XML elements [9] and this grouping process involves finding XML documents of the same schema and grouping them together.

3. Elements from different documents with similar structures can be stored together, resulting in reduced storage and faster query processing.

4. The important prerequisite for XML schema based classification/clustering of XML documents are the schema based algorithms where the availability of schemas is very important.

5. In element level clustering similarity between elements of the two schemas is checked and the XML document schemas are used for clustering.

Processing XML documents based on their schemas may not be feasible in certain cases in practice, since some XML documents are schema-less. Most authoring environments need to read and process schemas in order to understand and enforce the content models of the document. If an XML document relies on default attribute values, at least part of the declaration must be processed in order to obtain the correct default values. In case of web applications, invalidated input from web requests is considered as one of the important vulnerability and it may be avoided by using a schema. The semantics associated with white space in element content differs from the semantics associated with white space in mixed content. Without a schema, there is no way for the processor to distinguish between these cases, and all elements are effectively mixed content. The presence of a schema is important for automation and optimization of search engines. Schema is important for integrating (meta) data through schema matching, processing meaningful queries over XML data and in the area of generic model management.

Schema extraction is a very important step in the process of validation of heterogeneous XML document collection which contains documents belonging to different schemas and XML documents with no schemas exists. Data mining usually combines XML documents from a variety of sources with different schemas and considering the variety of the XML documents grouped and the complexity involved with it, it is a challenge to find good schemas automatically. Heterogeneous XML documents collection needs schema generation process, because each group of XML documents would have come from different schemas. Schema extraction is very useful in the implementation schema based classification/clustering algorithm for heterogeneous XML document collection. Many of the optimization procedures rely on a well-fitting schema to work properly and if there is no proper schema we would not be able to apply these optimization procedures. Extracting schema from a collection of XML documents has many real world applications and web applications that involve a lot of data. We can motivate a myriad of potential applications for efficient, automated schema discovery tools because naive approaches fail to produce the automatic generation of meaningful schema from underlying data.

1.3 Applications of XML

XML is a highly suited language to represent richly structured information such as dictionary content and any type of publishing house to marketing materials. XML has well-structured storage of information, which enables innovative browsing interfaces. XML is also a fundamental component in many XML web services and it is used to store and exchange data in the internet environment that may include private messages of customers. XML is widely used in arbitrary data structures like in web services and servers. XML is also used to process data in many application programming interfaces (APIs) and in schema systems that are used in developing XML-based languages. A large number of Office-productivity tool like Microsoft’s ‘Office word’, ‘Open office’ and Apple’s ‘iWork’ use XML-based languages. XML is useful in dynamically storing and retrieving data in a variety of web applications. XML is used in applications where you need portability and in internet portability of web documents is very essential because of the rapid increase in information volume across the internet. XML provides interoperability between different business applications and so it is widely used in internet which results in large volumes of heterogeneous XML data, i.e., combination of XML documents conforming to different schemas.

Storing in XML format is useful when the data gathered on the web will be sent to applications on non-Windows platforms. A relational database matches data by using the most common characteristics found within the data set and the resulting groups of data are organized and this makes the data more organized. For example, a data set containing all the book information in a library can be grouped by the year each book was published and so on. For such a grouping we need to sort the XML documents based on their schema. Hence, this process is useful in relational database management system (RDBMS). RDBMS has many applications in XML data storage in different fields like financial records, medical records, personal information, and manufacturing logistical data.

1.4 Contributions

In this thesis, efficiently extracting schema information from a large collection of XML documents will be studied. We propose a practical XML schema extraction method for a collection of XML documents by integrating some of the best XML schema extraction methods that are in practice. A system was proposed and implemented to address two key issues that arise in the extraction of schema from a collection of XML documents: accuracy and generalization of the inferred XML schema. It is very important to find tradeoffs between these two factors in order to produce the optimal XML schema. When we increase the accuracy of the schema generation process, the complexity and the size of the XML schema increases accordingly. On the other hand, when we make the schema more generalized, it may produce an XML schema that matches a broad range of XML documents. In this thesis, we will introduce a frequency factor to control the level of accuracy and the level of generalization of the target XML schema.

The minimum description length (MDL) computation is necessary for ranking the regular expressions. We propose the method of using the cost of a NFA to compute the MDL. We compare the MDL computation by using a NFA and by using regular expression. Based on these comparison result fine tunings to our techniques are made. Using NFA for finding the cost of regular expressions increases the accuracy and simplifies the matching process of the regular expressions. The effectiveness of this approach is increased when the XML files from fairly different XML data collections are used. The framework to compute edit distances between XML documents and inferred schemas provides an alternative to the traditional algorithms for tree-to-tree edit distance in evaluating structural similarity between XML documents [9]. The XML schema extraction approach presented in this thesis has several advantages over existing techniques. We will demonstrate the efficiency of our approach by using a sample XML data generator to produce sample input data from a given schema and regenerate the schema back from the sample input data by our approach. The sample inputs from the XML data generator are then mixed with noise and the schema is regenerated. The result from this experiment is then used for mapping with the input schema to determine the measure of similarity. We generated various other XML test cases like using a small number of large file XML files and a large number of small XML files. Synthesized XML data from various complex schemas over the internet are used for schema extraction and the results of these test cases show that our schema extraction technique is more accurate than other approaches and is many times faster than existing approaches.

1.5 Thesis structure

The remainder of this thesis is organized as follows. Chapter 2 presents the existing and works on XML schema extraction. Chapter 3 presents the proposed system, the techniques used in our system. Chapter 4 presents the results of our experimental studies, demonstrating the efficiency and effectiveness of our algorithms. A brief conclusion of the thesis is given in Chapter 5.

CHAPTER 2: LITERATURE REVIEW

XML schema is used to describe the XML documents and when schema is not available, schema information can be inferred from the existing XML documents. In practice, we need to automate the extraction of schema from a collection of XML documents to quicken this process of validation. The schema extraction often requires certain generalization of input data, because representing all the data as such will result in bulky data. The ideal extracted schema should tightly represent the data and on the other hand be concise and compact as well. So, automation of schema extraction becomes a difficult and challenging task. The regular expressions which are generated by using learning algorithms designed for deterministic finite automata (DFA) [15] and then turning these automata into regular expressions tend to produce expressions which are rather unreadable from a human perspective.

We need to infer schema from a collection of XML documents, but inferring a schema from a large collection of XML documents is challenging because:

1. XML document set acquired on the Internet tend to have noise documents which can significantly degrade the accuracy of the inferred schema (even though there is nothing wrong with the inference algorithms).
2. The procedure to infer the schema is time consuming when the number of documents is large or the individual document is large in size.
3. There are no good criteria to evaluate the quality of the extracted schema.

```
<?xml version="1.0"?>
<catalog>
    <book>
    <title>Harry Potter</title>
    <author>J K.Rowling
    </author>
    <year>1998</year>
    <price>16.99</price>
  </book>
  <book>
    <title>XML</title>
    <author>Author A</author>
    <author>Author B</author>
    <author>Author C</author>
    <author>Author D</author>
    <year>2011</year>
    <price>10</price>
  </book>
  </catalog>
```

```
|- catalog (34)
   |- #text (0)
   |- book (14)
          |- #text (1)
          |- title (3)
                 |- #text (2)
          |- #text (4)
          |- author (6)
                 |- #text (5)
          |- #text (7)
          |- year (9)
                 |- #text (8)
          |- #text (10)
          |- price (12)
                 |- #text (11)
          |- #text (13)
   |- #text (15)
   |- book (32)
          |- #text (16)
          |- title (18)
                 |- #text (17)
          |- #text (19)
          |- author (21)
                 |- #text (20)
          |- #text (22)
          |- author (24)
                 |- #text (23)
          |- #text (25)
          |- year (27)
                 |- #text (26)
          |- #text (28)
          |- price (30)
                 |- #text (29)
          |- #text (31)
   |- #text (33)
```

Figure 2.1: Generated Catalog data from corresponding XML tree.

Practically the number of elements in a collection of XML documents may be very large and so it is difficult to extract the schema out of every document in the collection. In such cases, we choose a small set of elements that may possibly be a good representative of that collection. The process of selecting the representatives from a collection must be done carefully such that the size of the XML document is not too small or too large, as the structure of those documents may not be common in that collection. Among the remaining documents, we randomly select a small subset (representative) for schema extraction and then they are merged with the other representatives.

```
<?xml version="1.0"?>
<catalog>
        <book>
        <title>Harry Potter</title>
        <author>J K. Rowling</author>
        <year>1998</year>
        <price>16.99</price>
    </book>
    <book>
        <title>XML</title>
        <author>Author A</author>
        <author>Author B</author>
        <author>Author C</author>
        <author>Author D</author>
        <year>2011</year>
        <price>10</price>
    </book>
    </catalog>
```

```
|- catalog
    |- book
        |- title
        |- author
        |- year
        |- price
    |- book
        |- title
        |- author
        |- author
        |- year
        |- price
```

Figure 2.2: Catalog data and corresponding parser output.

Inferring XML schema from a collection of heterogeneous XML documents, involves selecting a particular set of XML files that best represents the whole collection of XML files and compare the strings generated from the XML parser that parses the set of XML files. The intermediate parsed set of strings from a sample XML document is represented here and the next step is the process of inferring regular expressions from strings. Several publications have been published to address the problem of inferring regular expressions from the set of strings which is equivalent to mining regular expression from strings and so it is essential to know the existing methods. The problem of inferring regular grammars from examples has been studied in various systems like Xtract and IBM alphaWorks DDbE product. Among these systems, Xtract [6] provides a more practical solution for pattern extraction from a collection of string. One of the important contributions made by Xtract is the introduction of MDL in ranking candidate expressions and it concentrates on improving the efficiency in computing the MDL and adapts the framework for schema extraction for a large collection of XML documents. In general, the MDL principle states that the best grammar (schema) to infer from a set of data is the one that minimizes the sum of the length of the grammar and the length of the data when it is encoded with the grammar. Inferring regular expressions from a set of strings is used in sequence profiling [7], information extraction theory, and web structure mining [6].

Schema Extraction Process

XML schema has syntax that incorporates the full expressing power of regular expressions, manually deducing such a schema for even a small set of XML documents created by a user could prove to be a complicated process. A schema extraction algorithm should produce accurate and meaningful schemas that are also intuitive and appealing to humans (i.e., resemble what a human expert is likely to come up with). The Xtract [6] algorithm integrates many of the other schema extraction proposals as its subroutines and finally tries to find the best schema extraction technique out of these proposals. The generation of candidate schemas is an important task and is done in generalization. The three major steps in Xtract algorithm namely generalization, factoring and ranking are described here.

Generalization

The generalization component of this system infers a number of regular expressions which have been found frequently on the real life schemas. This step prevents the system to infer certain complex expressions that are less likely to occur in practice. If the number of sequences that we consider for generalization is very less we may over generalize the resulting schema and if the number of sequences is very large it needs to be more generalized. This step is useful for generating the candidate schemas for the calculation of the minimum length description process. In real life there may be numerous candidate schemas that generalize the sub element sequences in the input, and choosing the schema that best describes the structure of these sequences is a non-trivial task.

The generalization process involves two important steps, namely finding the 'sequential' pattern and the 'Or' patterns. The first step is the process of discovering the sequential patterns and it involve setting up of the input threshold parameter value to r>1, which is the minimum number of contiguous repetitions of subsequence r in s required for the repetitions to be replaced. The maximum numbers of repetitions are analyzed and the longest among them is chosen and the ties are subsequently resolved.

In the process of discovering 'Or' patterns, the first step is the process of procedure partition where the 'Or' patterns are discovered based on the various values of *d* to control the degree of generalization. We are using the novel heuristic algorithms for finding patterns in each input sequence and replacing them with appropriate regular expressions to produce more general candidate schemas. Kleene stars generated are used to produce regular sub expressions that generalize the patterns observed in the input sequences. The generalization rules that we use are based on the discovery of frequent, neighboring occurrences of subsequences and symbols within each input sequence. The explosion in the number of resulting patterns is controlled by practical, real-life schema examples as used in Xtract algorithm. Although simple in most cases, the heuristics are effective for real data sets.

Factoring

The factors of the common sub-expressions from the generalized candidate schemas obtained from the generalization step need to be more concise. For example if there is a simple expression like *(ab+ac)*, it can be further factored to *a(b+c)*. By doing this we are reducing the cost of encoding of the schema, as we can see the size is considerably reduced. This reduction of cost will result in more efficient schema generation. There are several choices made while factoring a common real life schema based on the sequences that are on the schema and these factored forms of the candidate schemas must also be added to the MDL calculation module. Even though all schemas may further be factored, not all schema factoring are efficient. We are using tight rules are factoring the data and the cost of factoring may also be a factor in improving the efficiency of schema generation and so, we try to generate different schema files and try to find the optimum method of schema generation. The MDL principle is the one that is general enough to cover a large subset of the input sequences, but at the same time the structure of the large subset is also brought out. We are using the greedy algorithm to share the common prefixes and suffixes and have the minimal overlap with other selected schemas.

Ranking

The final and the most important contribution of Xtract is the introduction of MDL to rank the candidate expressions. The MDL principle has its roots from the information theory and it provides an optimal theory for schema extraction. The ranking of the candidate schemas is done by choosing the best set of candidate schemas that best represent the original input conditions. Choosing the best schema may be done by computing the cost of each and every candidate schemas and ranking the schema based on the number of bits required to describe the schema. This ranking thus provides us a direct measure of the conciseness. Selecting the optimum schema from the collection of XML documents involves the natural mapping in to the Facility Location Problem (FLP), which is known to be NP-complete and so we designed a efficient approximation algorithm with guaranteed performance ratios.

A target function $L=L_s+L_d$ is used to deduce the optimal schema where,

L_s denotes the length of the schema, and

L_d denotes the length of the documents encoded using the corresponding schema.

The target function helps to give a measure of the optimum schema that can be generated from the candidate schemas that are generated from the previous steps. The sample patterns that may occur in a given collection of XML documents is shown here, which gives an overview of the kind of ranking algorithm that is being used.

Take the string *ababab* as an example: we may have the following three patterns:

1) *ababab*

2) (*ab*)*

3) (*a*|*b*)*

It is easy to see that pattern 1) is longest, but when used for encoding the input string, it would be shortest as the input is just one occurrence of the pattern. Pattern 3) can be the simplest, but when used in encoding the string, it is longest as there are six steps of choices. The MDL principle gives the best theory to infer from a set of data is the one which minimizes the sum of the length of the theory in bits and the length of the data in bits when encoded with the help of the theory.   Each schema is assigned a MDL cost and the schemas with the lower MDL costs are ranked higher. Based on these ranks the given XML files are used to determine the optimum schema.

CHAPTER 3: PROPOSED SCHEMA EXTRACTION PROCESS

Although inferring schema from an XML document is simple to describe, inferring a schema from a large collection of XML documents is a challenging task. We need to balance the precision and generality factor of the extracted schema from a XML document collection. We need to make sure that the extracted pattern precisely describes the sample documents without covering those documents (with different structures) outside of the sample set. At the same time the extracted pattern should be general enough to cover those documents that share similar structure with those in the sample set. According to our proposed algorithm, the frequencies of the child sequences are evaluated and the sequences with low frequencies may not be covered by the inferred grammar. This would help reduce the negative effects of noises in the classification process. The minimization function used in Xtract is modified by introducing a frequency factor that helps to maintain the balance between the generalization and specialization of the schema being generated.

We employ Rissanen's MDL principle [16] to derive the optimal schema from the set of candidate schemas. To balance the precision and generalness of the inferred schema, a frequency factor (λ), is introduced in the target function for the MDL. This frequency factor achieves the balance between the generalization and accuracy of the generated schema because it allows the user to set the optimum range value to find a schema almost close to the original schema. The selection of the optimal schema from our algorithm has a direct and natural mapping to the Facility Location Problem (FLP) which involves the evaluation of various sites for a new facility. We are using a NFA in order to match the existing candidate schemas to find more generalized schema. The step has simplified the process of generating the more generalized schemas and some of the complex schemas that cannot be generated from the traditional Xtract algorithm can be generated by the use of NFA in the factoring step. The improvements made in our schema extraction system from a collection of XML documents are clearly described below.

3.1 Frequency Factor (λ)

The first heuristics is the introduction of a balancing factor λ for the target function in calculating MDL. Whenever a pattern is to be extracted from a collection of XML documents, there are the following important two factors:

Precision: The extracted pattern precisely describes the sample documents while doesn’t cover those documents (with different structures) outside of the sample set.

Generality: The extracted pattern should be general enough to cover those documents that share similar structure with those in the sample set.

Instead of using the target function $L=L_s+L_d$, $L=\lambda L_s+L_d$ is used as a target function for our proposed system for the overall cost minimization. The parameter of λ is used to balance the precision and generality of the inferred schema.

A target function $L= \lambda L_g + L_d$ is used to deduce the optimal schema where,

λ denotes the frequency factor,

L_s denotes the length of the schema, and

L_d denotes the length of the documents encoded using the corresponding schema.

The introduction of λ is used to control the ranking of different patterns that are generated for schema extraction. When the same sequence patterns may appear in the input documents multiple times, the frequency can be used to give the input with higher frequency more weight in the overall MDL calculation. So the optimization of $L=\lambda L_s+L_d$ can be represented as,

$$L=\lambda L_s+\sum \mu_i L_{di}$$

Where μ_i is the weight of sequence i. This optimization procedure not only improves the accuracy of the inferred schema, but also reduces the time needed for inference of schema.

3.2 NFA for computing MDL

The second improvement is the introduction of NFA for regular expression matching in computing the MDL. We compute the cost of encoding a sequence using a NFA simulation of the regular expression, instead of computing the cost of encoding a sequence using a regular expression. The cost to encode a string using a regular expression can be computed recursively [6]. For an input string of length *l*, it take *O(l*$\times$*s)* time to compute the cost of encoding where s is the size of the input document. When the size of the input document is large, this can be very time consuming. We will show this can be reduced to *O(l)* in practice when NFA simulation is used.

An NFA is a 5-tuple $A = \langle \Sigma, Q, Q_0, F, \delta \rangle$ structure where,

Σ is the finite set of input symbols,

Q is a finite set of states,

$Q_0 \subseteq Q$ is the set of starting states,

$F \subseteq Q$ is the set of final states,

δ is the transition function defined from $pow(Q) \times \Sigma \cup \{\varepsilon\}$ to $pow(Q)$, and

$pow(Q)$ is the power set of Q.

The NFA with ε-moves (also sometimes called NFA-epsilon or NFA-lambda) replaces the transition function with one that allows the empty string ε as a possible input. We could also show that ordinary NFA and NFA with epsilon moves are equivalent and that one can construct the other, which recognizes the same language.

NFA Construction

Thompson's NFA construction algorithm may be used for converting a regular expression in to an NFA. This method constructs a regular expression from its components using epsilon transaction (ε) concept. The epsilon transaction (ε) acts as glue for the subcomponent NFA's. A ε-transaction adds nothing because there may be no change to the original regular expression, but still it is very useful. An NFA can be constructed from a regular expression using Thompson's method.

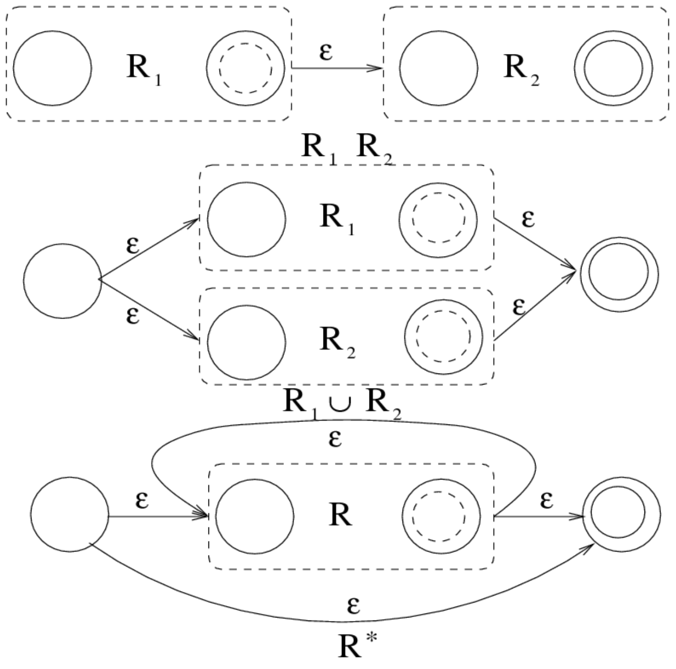


Figure 3.1: Thompson's method for NFA construction.

Let us construct an NFA for the regular expression ab^*+b. Now, comparing this to the form r_1+r_2, where $r_1=ab^*$ and $r_2=b$, we can construct the NFA for this specific regular expression. We will construct the automation for r_1, which in turn can be represented by $r_1=r_3r_4$, where $r_3=a$ and $r_4=b^*$. The automation for r_3 and r_4 can be represented by the following figure.

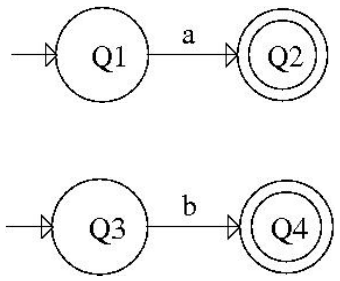


Figure 3.2: NFA for $r_3=a$ and $r_5=b$ in $r_4=r_5^*$.

The NFA for the regular expression r_2 is simple and can be represented by constructing the respective automation for r_2 as shown in this figure.

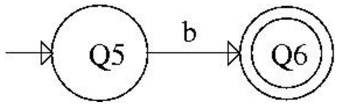


Figure 3.3: NFA for $r_2=b$.

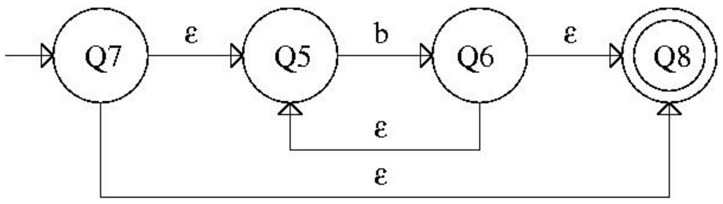


Figure 3.4: NFA for $r_4=b^*$.

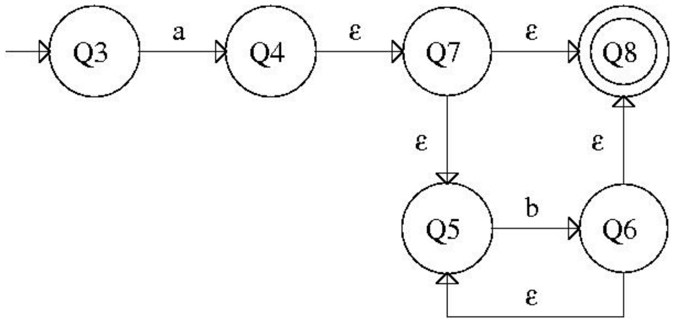


Figure 3.5: NFA for $r_1=ab^*$.

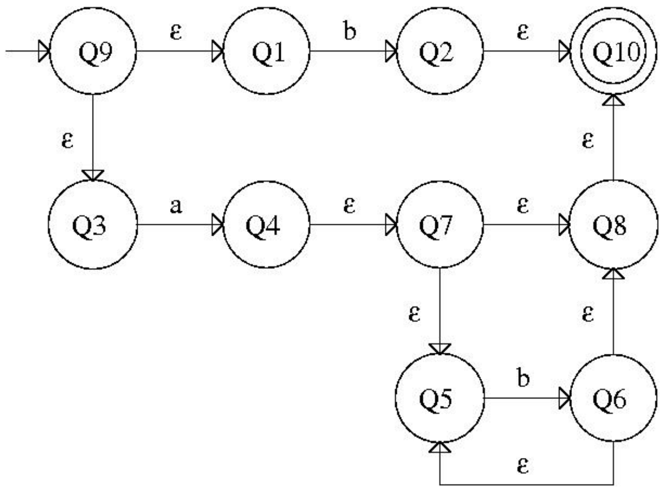


Figure 3.6: NFA for $r_4=ab^*+b$.

NFA engine processes a particular language element by using greedy matching that matches as much of the input string as possible also saves its state after successfully matching a sub expression. If a match eventually fails, the engine can return to a saved state so it can try additional matches and this process of abandoning a successful sub expression matches so that later language elements in the regular expression can also match is known as backtracking. NFA engines uses backtracking to test all possible expansions of a regular expression in a specific order and accept the first match. Since a traditional NFA backtracks, it can visit the same state multiple times if it arrives at the state over different paths. So, it can run exponentially slowly in the worst case. Because a traditional NFA engine accepts the first match it finds, it can also leave other (possibly longer) matches undiscovered. Traditional NFA engines are more favoured by programmers because they offer greater control over string matching than either DFA or POSIX NFA engines they can run slowly in the worst case, you can steer them to find matches in linear or polynomial time by using patterns that reduce ambiguities and limit backtracking. In other words, although NFA engines trade performance for power and flexibility, in most cases they offer good to acceptable performance if a regular expression is well-written and avoids cases in which backtracking degrades performance exponentially.

Lazy construction method is used when we encounter a large XML document that cannot be computed directly by using a NFA. Some experiments are done to evaluate the space usage of the lazy construction to make sure that the method is feasible for large document in real applications and as a result of these experiments, DFA in place of NFA will not significantly introduce more space usage, and using the lazy construction to get

DFA for matching cost is feasible for large XML documents. The performance of NFA simulation can be significantly improved by using a lazy construction of DFA. Although the size of a DFA can be exponential to the size of the corresponding NFA, the size of the DFA in practice tends to be small. In fact, the number of states in the DFA is smaller than the size of corresponding NFA in many cases.

NFA cost computation

Given an NFA and an input string, we can determine whether or not the NFA accepts the input by using the NFA simulation algorithm. NFA simulation algorithm basically matches the given input string with the NFA that is being considered and once when the given word is accepted the cost of the word is sent as a result. If the given string did not match the NFA, then it returns -1. It is a simple algorithm and very useful for calculations.

```
// computes  the cost of NFA simulation
// returns -1 if the input is not accepts by the NFA
int accept(NFA nfa, String input) {
    int cost = 0;
    Vector current = new Vector(nfa.numOfStates());
    current.set(nfa.start, true);
    for (each literal in input) {
        current = transit(current, literal);
        cost = cost + countTrue(current);
    }
    if(current.get(nfa.final))
        cost = -1;
    return cost;
}


Vector transit(Vector current, int literal)
{
    Vector next = new Vector(current.size());
    for (each entry in current)
        if (current.get(i) == true)
            for(each j s.t. $j \in \delta(i, literal)$)
                next.set(j, true);
}
```

Figure 3.7: NFA simulation algorithm.

NFA Performance Evaluation

Although the non-determinism in an NFA is not preferred in many applications, it provides a similar measure as encoding an input string using a regular expression. As the non-determinism in a simulation cost increases, it will result in longer corresponding encoding using the regular expression. It takes $O(ls)$ time to use NFA simulation to determine whether or not an input string can be accepted by an NFA, where l is the length of the input string and s is the number of states in the NFA. When we use a DFA, it contains only finite state and so it cannot match a pattern with back references. It does not construct an explicit expansion and so, it cannot capture sub expressions. When traditional NFA engines perform pattern matching, their processing order is driven by the regular expression pattern.

3.3 Introducing edit distance concept

The third contribution of this thesis is the introduction of edit distance to quantify the quality of the schema. We use the edit distance between XML document and schema (denoted by $\delta(d,s)$, where d is the document and s is the schema) [11], which is defined as the minimum cost of the edit scripts (insert a node, delete a node, or replace a node) that transforms an XML document such that it conforms to the XML schema. As the edit distance is related with the size of the document, we use normalized distance concept.

$$\hat{\delta}(d,s) = \frac{\delta(d,s)}{|d| + |s|}$$

It is essential for us to measure for how well the document *d* conforms to schema *s*. Consider the case of clustering the XML data. Here, there are two types of algorithms for specifying XML documents similarities. First are structure-based only algorithms and second are structure and content algorithms [19]. Generally the main difference between these two methods is that element level specifies the similarity between tree elements (particularly the similarity in the name structure) whereas structure level studies the similarity of the whole tree and ignores the elements details [21].

Among these Pair wise based algorithms are more common which first create a similarity matrix for each pair of documents. This matrix is initialized by a criterion for measuring similarity between two documents. The pair wise algorithms use tree edit distance to calculate the similarity between two XML documents. We need to calculate the minimum distance between two trees T_1 and T_2 while using insert, delete and edit operations for each node of the tree. We must consider a cost for each of these operations that are computed depending on the node tags. So in order to specify the similarity between two XML documents, it suffices to calculate the minimum cost of converting T_1 tree to T_2 tree by a sequence of operations [21]. In other words, we calculate how many operations are required to convert T_1 to T_2. This cost is the similarity between T_1 and T_2.

The edit distances depend on the size of the document, as a larger document tends to have larger distance to a schema even if it conforms to the schema better than a small document. To overcome this, we use normalized edit distance. For a document *doc* and a schema *r*, the normalized edit distance can be defined by the following formula:

$$\hat{d}(r, doc) = \frac{C[v_S, F[1..|doc|]]}{|doc| + \min(r)}$$

where $|doc|$ denotes the size of the document, and $\min(r)$ denotes the size of the smallest document that can be generated by the schema r from the starting symbol v_s in r. Based on the definition of $C[v_{S,} F[1..|doc|]]$, the normalized edit distance ranges between 0 and 1, where 0 means the document *doc* conforms to schema *r*, and 1 means a complete deletion of *doc* and a rebuilding of a new document from *r*.

Given a document doc_i, a collection of documents *D*, and the corresponding schema $s(D)$ for *D*, we define the distance between doc_i and *D* as

$$\delta_{i,D} = \hat{d}(s(D), doc_i)$$

Similarly, the pair-wise distance of documents d_i and d_j is defined as

$$\delta(doc_i, doc_j) = \frac{(\delta_{i,\{doc_j\}} + \delta_{j,\{doc_i\}})}{2}.$$

The value of $\delta(doc_i, doc_j)$ lies between 0 and 1. A smaller $\delta(doc_i, doc_j)$ indicates the two documents being structurally more similar. In the case of a document clustering task, a matrix of pair-wise distances may be used to feed the clustering algorithm.

Generally the main differences between pair wise and incremental algorithms are time complexity and application. In the best case, pair wise algorithms are from order of two but incremental algorithms work in linear time and are useful for online environments [19]. Hence, we can use this edit distance concept for our schema computation. In our algorithm, given a collection of XML documents with similar structures, an XML schema can be inferred from training documents from the collection. The quality of the inferred schema can be measured by the normalized edit distances between the testing XML documents and the schema.

CHAPTER 4: IMPLEMENTATION AND EXPERIMENTAL STUDIES

In this section, we investigate the efficiency, effectiveness, and scalability of the algorithms presented in this chapter on various data sets. The general overview of our experiment to be conducted is shown in the figure.

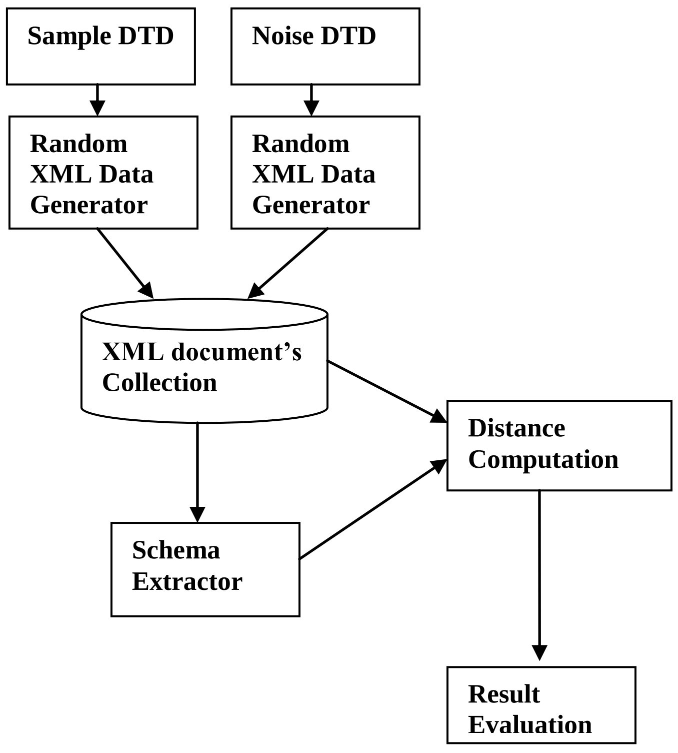

Sample DTD
Noise DTD
Random XML Data Generator
Random XML Data Generator
XML document's Collection
Distance Computation
Schema Extractor
Result Evaluation

Figure 4.1: Evaluation framework.

The data sets to be used in our experiments are generated by an XML generator. The input DTD and the noise DTD are given as inputs to the XML data generator which generates the XML documents conform to the input DTD and the noise DTD respectively. These XML documents collections are then used as inputs for the schema generation algorithm. The distance between each XML document is calculated by MDL principle of our proposed XML schema extraction algorithm and eventually we are generating the schema corresponding to the particular set of XML documents. Result evaluation is the process of showing the effectiveness and efficiency of our algorithms by comparing our results with previous existing algorithms.

4.1 Overview of the implementation

Eclipse is the Integrated Development Environment (IDE) that we use for the java implementation of the algorithms in this thesis. The experiments were performed on a laptop having Intel Core2 Duo-2.2 GHz processor running Microsoft-Windows 7 platform and four Giga Bytes of main memory running eclipse. We used IBM's XML generator jar file [13] and it was implemented using a java program to generate the random XML files from the DTDs that are edited using WordPad.

The proposed method was implemented in Java and we are using the XML4J parser1 to parse XML documents. We are implementing both the original algorithm and our modified algorithm in Java to make a good comparison of improvement in efficiency by using our method. We implemented the MDL Subsystem using the greedy method instead of the Facility Location Problem (FLP) approximation [6]. Implementation of the schema extraction algorithm may consume less time to extract the schema than other algorithms because of using NFA that is faster than the regular expression implementation. The implementation involves a regular expression class that helps to convert the regular expression that was used to compared before in to individual NFA's and then calculate the distance value from the resulting NFA's. The edit distance normalization that we used is known to improve the processing of multiple XML documents, as this concept is widely used in XML cluster analysis for this purpose. XML documents are represented as ordered, labeled trees and the tree edit distance is used to find the similar XML documents. We also used frequency factor that we were talking about in the previous chapters in our implementation of the proposed algorithm. The frequency is fixed to a variable and depending on the results it is altered many times, so that we can come to an optimum frequency value that balances the generalization and specialization of the resulting schema for the most common XML document collection that may be encountered in the practical world. In order to show the improvement in efficiency and accuracy, we compare the resulting schemas of our approach with results from some of the previously implemented algorithms.

4.2 Test data

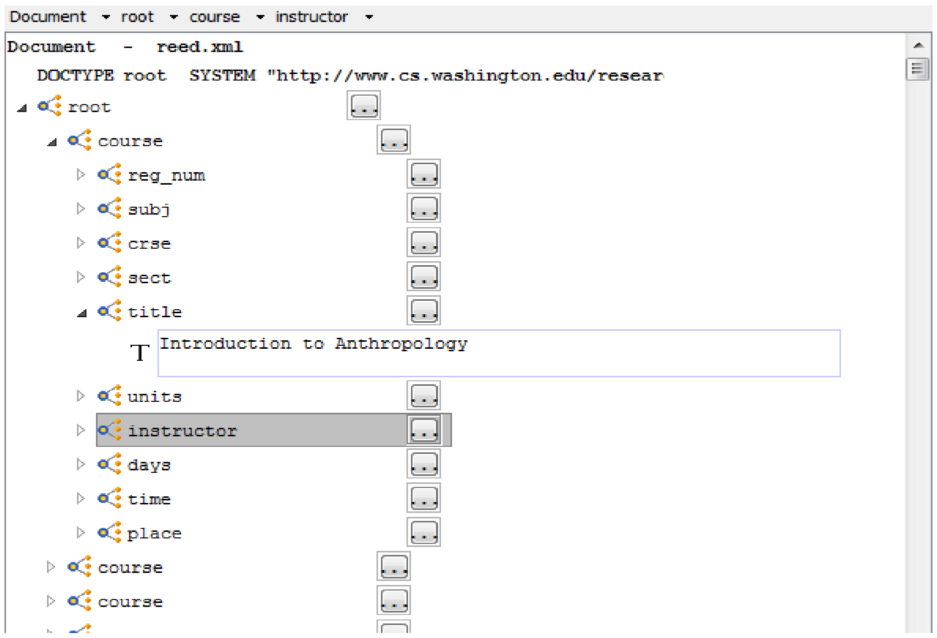
A java program to generate the random XML data is written by using IBM's XML generator jar file. Our test data is generated by using the sample input DTDs. We can control the number of elements in that particular XML file and the size and various other fine details in our program. The generation of data set is a very important part of testing and further development of our proposed system because, we design our proposed system based on the results produced. Careful designing of the data set is required in order to make our testing more meaningful. All the experiments in this paper were done on synthetic XML documents generated manually derived from [22, 23]. The XML document collection data set contains 500 XML documents including a small frequency of noise which is important to make the collection of XML documents heterogeneous and test the frequency factor that we use. The number of tags varies from 10 to 200 in these sources and the nesting level varies from 2 to 30. Majority of these documents contains a number of different documents that have structural and semantic differences. Hence, even though documents are from the same DTD, they might not be considered similar enough to be grouped into the same type of XML document [20].

```
        <!ELEMENT inproceedings (( author, ( chapter| volume| note| publisher| month|
ee, publisher| pages, journal) | cite, ( note| journal| note, title| author, journal| editor, url|
publisher, isbn) | year, ( month| url| year| crossref, isbn| month, school) | cdrom, ( isbn|
booktitle| year| chapter, booktitle| crossref, title) | ( publisher| pages| publisher, volume|
volume| note, cdrom) , author| month, ( ee| school| cite, series| publisher, chapter) |
editor, ( url| pages| cite, title| address, address) | ( school| title| volume| url, school) ,
number| booktitle, ( journal| booktitle| volume, url) | note, ( year| year, title| month,
cite) | ( url| title| crossref) , pages| series, ( series| number, title) | journal, ( series| url,
address) | isbn, ( journal| address, year) | address, ( note, month| year, isbn) | author| ee|
url| editor| EMPTY| series| volume| cdrom| number| month| journal| school| address|
cite| crossref| note| isbn| pages| ee, journal| crossref, ee| volume, volume| school,
publisher| chapter, note| pages, url| number, isbn, booktitle) ) >
        <!ELEMENT publisher (EMPTY) >
        <!ELEMENT volume (EMPTY) >
        <!ELEMENT author (EMPTY) >
        <!ELEMENT cdrom (EMPTY) >

        <!ELEMENT chapter (EMPTY) >

        <!ELEMENT booktitle (EMPTY) >

        <!ELEMENT ee (EMPTY) >

        <!ELEMENT pages (EMPTY) >

        <!ELEMENT journal (EMPTY) >

        <!ELEMENT series (EMPTY) >

        <!ELEMENT cite (EMPTY) >

        <!ELEMENT note (EMPTY) >

        <!ELEMENT title (( ref, ( sub, i| sup| sup, tt) | sub, ( ref| ref, i) | ( tt| sup, i) , sub|
EMPTY| sup| sub| i| i, i) ) >

        <!ELEMENT sub (( sub, ( tt, ( sub| ref) | sup| i| sub, i) | tt, ( ref| sup| ref, ref| sup,
sub) | i, ( ref| sup, ref| ref, sup| i) | ( ref| sup, i) , tt| sup| tt| i| ( sub| ref) * ) ) >

        <!ELEMENT sup (( sub, ( tt| ref| i, tt| sup, i| sub) | ref, ( ( sub| i) , sub| sup| ref, i) |
i, ( sup| ref, sup) | tt| i| sup| sub| tt, ref| sup, tt, sub| ref* ) ) >

        <!ELEMENT tt (( tt, ( ref| tt| i| sub| tt, sup) | i, ( tt| ref| sub, sup| ref, sub) | sub, ( i|
tt| i, sup| tt, tt) | ref, ( sup| ref, sub) | tt| ref| i| sup, ref, sub| ( sub| sup) * ) ) >

        <!ELEMENT i (( ref| sub| i| sup| ref, sub| tt* | tt, sup, sup| i, sup, i| ( tt, ( sub| tt) |
ref| i| sup| tt) , tt| ( ref| i) , sub, ref) ) >

        <!ELEMENT ref (EMPTY) >

        <!ELEMENT year (EMPTY) >

        <!ELEMENT month (EMPTY) >

        <!ELEMENT url (EMPTY) >

        <!ELEMENT editor (EMPTY) >

        <!ELEMENT number (EMPTY) >

        <!ELEMENT crossref (EMPTY) >

        <!ELEMENT school (EMPTY) >

        <!ELEMENT address (EMPTY) >
        <!ELEMENT isbn (EMPTY) >
        <!ELEMENT book (( title, ( journal| year| month| editor, note| note, url) | ( note|
year, pages| crossref, cite| editor, crossref| number, url) , publisher| ( isbn| url| chapter,
editor| year, year| pages, author) , chapter| volume, ( month| number| title| pages) | (
isbn| year, year| note, cdrom| isbn, author) , volume| author, ( author| editor| year, year)
| url, ( isbn| booktitle| pages, ee) | cdrom, ( school| url, crossref| booktitle, url) | ee, (
series| journal, note| cdrom, isbn) | month, ( year| series, title| year, school) | note, (
journal| year, author) | school, ( address| crossref, booktitle) | booktitle, ( url| booktitle,
pages) | crossref, ( number, editor| volume, cdrom) | EMPTY| note| number| url| school|
cite| year| editor| title| author| ee| pages| journal| cdrom| booktitle| publisher, journal|
series, pages| pages, number| chapter, title| number, year| editor, chapter, url| isbn,
number, address| cite, journal, ( volume| note) ) ) >
```

Figure 4.2: Generated output DTD.

XML documents are also generated from dblp DTD [24] and used it to test the accuracy of our method. In order to make the structural and semantic differences between the XML documents generated, we split this DTD in to four sub DTDs. We created these sub-DTDs in a way that the derived synthetic XML documents would share almost the same node tags in most of the levels, but may also contain some different relationships in some of the levels. We designed the first sub DTD such that, it contained only the two nodes namely, "article" and "in proceedings" node in the first level of the dblp DTD and the corresponding nodes in the sublevels. The second sub-DTD contained the two nodes namely "proceedings" and "book" node. The third sub-DTD contained only the "book" node and the last sub DTD contained the “master thesis” and “www” node. The generated XML data set by using XML generator consists of a total of 500 XML files. 100 XML files derived from the original DTD, 100 XML files derived from the first sub-DTD, 100 XML files derived from the second sub-DTD, 100 XML files derived from the third sub-DTD, and 100 XML files derived from the fourth sub-DTD are combined to form this collection of 500 XML files. The test case is created by using these different sub-DTDs because it avoids the redundant data sets from the XML data generator. The elimination of redundancy is very important when we try to create a test case that is a close match of a practical XML document collection. The output DTD in the figure denotes the DTD generated from algorithm by using the test case described here.

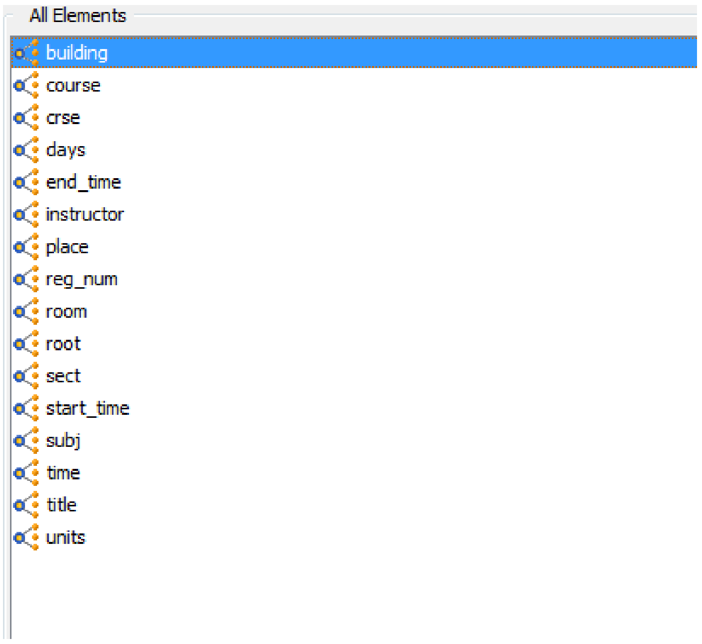
Some of the real-life DTDs are used in our experiments to find the accuracy of our proposed algorithm. We generated 200 to 1000 XML documents for each DTD document using the XML Generator and used it for our study. The important aim of our test is to make sure that our method performs well in most common practical situations of XML schema extraction and so we selected the most common DTDs that are available on the internet. For instance in one of our test case, we generated a 500 XML files belonging to the same DTD and mixed them with 10 noise files of different DTD in order to simulate a more common test case.



Document ▾ root ▾ course ▾ instructor ▾

```
Document  -  reed.xml
   DOCTYPE root  SYSTEM "http://www.cs.washington.edu/resear
◢ root
   ◢ course
      ▷ reg_num
      ▷ subj
      ▷ crse
      ▷ sect
      ◢ title
            T Introduction to Anthropology
      ▷ units
      ▷ instructor
      ▷ days
      ▷ time
      ▷ place
   ▷ course
   ▷ course
```

Figure 4.3: Synthesized XML document.



All Elements

- building
- course
- crse
- days
- end_time
- instructor
- place
- reg_num
- room
- root
- sect
- start_time
- subj
- time
- title
- units

Figure 4.4: Synthesized XML DTD elements.

4.3 Experimental studies on efficiency

No of files	**Noise**	**Time with DFA**	**Time without DFA**
200	1	1164	1225
200	3	8535	8609
200	2	41101	42959
200	20	5254	5398
200	10	5052	5533
500	10	22570	23033
500	30	26670	28643
500	50	30004	31964
450	50	26330	26810

Table 4.1: Comparing time taken for schema inference.

The second group of study is to evaluate the improvements in running time. Please note that the lazy construction of DFA works better when the number of sample input and the size of the input are large enough. The comparison of the time for schema inference from different xml documents using original regular expression matching and our proposed method is presented in the following table. Please note the file size is not the exact, and unit for the time in the table is millisecond.

The time taken for the generation of the DTD is calculated by finding the difference between the system time at the point of starting the execution of the program and the end of the execution of the program. We are comparing the total time taken to compute the MDL by using NFA, with the total time taken without NFA. The total time taken to compute the MDL by using NFA is 121022339 milliseconds and by using regular expression are 248041995 milliseconds for this set of data. From this sample data the time improvement factor (f) may be calculated which denotes the time improvement obtained from our proposed method of schema extraction from XML document collection. The data shown above is one of the actual data obtained from our java program implementation that used the sample XML documents that we generated for our study. The improvement factor in this case is found to be around 2 and so we can say by using NFA for the schema extraction process is approximately two times faster. The above comparison clearly shows that our method reduces the time needed to infer schemas from a collection of XML documents.

$$F = \frac{\text{time without NFA}}{\text{time with NFA}} = \frac{248041995}{121022339} = 2.05$$

4.4 Experimental studies on the quality of schema

```
<!ELEMENT #text () >
<!ELEMENT personnel (person* ) >
<!ELEMENT person (name, email* , ( url* | url* , link) ) >
<!ELEMENT name (( family, given| given, family) ) >
<!ELEMENT given (EMPTY) >
<!ELEMENT family (EMPTY) >
<!ELEMENT url (EMPTY) >
<!ELEMENT email (EMPTY) >
<!ELEMENT link (EMPTY) >
<!ELEMENT PARTS (TITLE, PART* ) >
<!ELEMENT TITLE (EMPTY) >
<!ELEMENT PART (ITEM, MANUFACTURER, MODEL, COST) >
<!ELEMENT ITEM (EMPTY) >
<!ELEMENT MANUFACTURER (EMPTY) >
<!ELEMENT MODEL (EMPTY) >
<!ELEMENT COST (EMPTY) >
```

Figure 4.5: Input DTD.

XML files with different complexities are given as input and the results on quality check show that the DTDs generated from our algorithm to be closest to the optimal DTD. The input DTD used for XML generation here are obtained from practical sources over the internet. We are selecting a set of XML files that does not have much of duplicate data and with fair amount uniqueness among the set of XML files generated from our random XML data generator.

The input DTD with moderate complexity level is used in this experiment and our random XML data generator generates the sample XML document collection and a total of distinctly different 500 random XML files are generated by using the input DTD shown here. The XML files generated differs in structure and size and the files size of the XML files generated ranges from 50 kilo bytes to 50 mega bytes. The 500 XML files are used as input along with 20 XML files with different schemas (noise data) to the samples. In validation testing, the first 200 documents are used as input and about 20 documents with different schemas (noise data) were inserted to get the test case. The remaining 300 documents are used to validate the schema inferred from the samples. The DTD is generated from our implementation of the proposed algorithm and the quality of the DTD generated may be calculated by mapping the DTD generated to the input DTD.

According to our experiments with different DTD samples, we realized that our output depends on the frequency of occurrence of a particular DTD type and it becomes clear when we see the following test case, where the input gets changed considerably with the change in frequency of a set of XML data. The XML files in different collections of XML documents are considerably altered, by changing the frequency of occurrence of one or more of the certain pattern in DTD file.

```
        <!ELEMENT personnel (( person, ( person| person, person) | person) ) >
        <!ELEMENT person (( ( name, email| name) , ( ( email, ( email| url| email, url) |
email| url) , ( url, ( link| url| url, link) | link| url) | email| url| link| email, email) | name) )
>
        <!ELEMENT name (( family, given| given, family) ) >
        <!ELEMENT given (EMPTY) >
        <!ELEMENT family (EMPTY) >
        <!ELEMENT url (EMPTY) >
        <!ELEMENT email (EMPTY) >
        <!ELEMENT link (EMPTY) >
        <!ELEMENT PARTS (TITLE, PART, PART, PART, PART) >
        <!ELEMENT TITLE (EMPTY) >
        <!ELEMENT PART (ITEM, MANUFACTURER, MODEL, COST) >
        <!ELEMENT ITEM (EMPTY) >
        <!ELEMENT MANUFACTURER (EMPTY) >
        <!ELEMENT MODEL (EMPTY) >
        <!ELEMENT COST (EMPTY) >
```

Figure 4.6: Generated DTD from collection-1.

```
        <!ELEMENT personnel (( person, ( person| person, person) | person) ) >
        <!ELEMENT person (( name, ( ( email* | email* , url) , ( url| url, url) | email, (
email| email, email) | email| email* , url* , link) | name| name, email, email) ) >
        <!ELEMENT name (( family, given| given, family) ) >
        <!ELEMENT given (EMPTY) >
```

```
<!ELEMENT family (EMPTY) >

<!ELEMENT url (EMPTY) >

<!ELEMENT email (EMPTY) >

<!ELEMENT link (EMPTY) >

<!ELEMENT PARTS (TITLE, PART, PART, PART, PART) >

<!ELEMENT TITLE (EMPTY) >

<!ELEMENT PART (ITEM, MANUFACTURER, MODEL, COST) >

<!ELEMENT ITEM (EMPTY) >

<!ELEMENT MANUFACTURER (EMPTY) >

<!ELEMENT MODEL (EMPTY) >

<!ELEMENT COST (EMPTY) >
```

Figure 4.7: Generated DTD from collection-2.

```
<!ELEMENT personnel (person) >

<!ELEMENT person (name) >

<!ELEMENT name (family, given) >

<!ELEMENT given (EMPTY) >

<!ELEMENT family (EMPTY) >

<!ELEMENT url (EMPTY) >

<!ELEMENT email (EMPTY) >

<!ELEMENT link (EMPTY) >

<!ELEMENT PARTS (TITLE, PART* ) >

<!ELEMENT TITLE (EMPTY) >

<!ELEMENT PART (ITEM, MANUFACTURER, MODEL, COST) >

<!ELEMENT ITEM (EMPTY) >
```

```
<!ELEMENT MANUFACTURER (EMPTY) >

<!ELEMENT MODEL (EMPTY) >

<!ELEMENT COST (EMPTY) >
```

Figure 4.8: Generated DTD from collection-3.

```
<!ELEMENT personnel (person) >

<!ELEMENT person (name) >

<!ELEMENT name (family, given) >

<!ELEMENT given (EMPTY) >

<!ELEMENT family (EMPTY) >

<!ELEMENT url (EMPTY) >

<!ELEMENT email (EMPTY) >

<!ELEMENT link (EMPTY) >

<!ELEMENT PARTS (TITLE, PART* ) >

<!ELEMENT TITLE (EMPTY) >

<!ELEMENT PART (ITEM, MANUFACTURER, MODEL, COST) >

<!ELEMENT ITEM (EMPTY) >

<!ELEMENT MANUFACTURER (EMPTY) >

<!ELEMENT MODEL (EMPTY) >

<!ELEMENT COST (EMPTY) >
```

Figure 4.9: Generated DTD from collection-4.

The input DTD represents the original DTD corresponding to the XML files collection that contains the four different collections of the sample XML files are taken for our experiments and the tests are done by decreasing and increasing the frequency of occurrence of a particular pattern of data in the DTD and the results are shown below. The results of our experiments shows that the frequency of occurrence of the type of XML files can considerably change the pattern of the DTD generated. Some of the data that we used for our experiments are shown here and the accuracy of the DTD that we compute through our implementation of our techniques is seen from these results.

4.5 Experimental studies on optimum frequency value

The fine tuning of the frequency and maintaining the balance between accuracy and generalization of the generated DTD documents are done by mapping the frequency of the DTD’s generated from our approach. For this purpose, we are carefully designing our test case such that mapping would be easy and efficient. For our experiments, we take two different DTD’s DTD-1 and DTD-2 that are distinctly different from one another. We first generate 500 random XML files corresponding to DTD-1 and DTD-2 and we split those 500 XML files corresponding to DTD-1 in to 5 groups of 100 each and similarly, we are splitting those 500 XML files corresponding to DTD-2 in to 5 groups of 100 each. 10 files from the first group of random XML files generated from DTD-1 is taken and similarly, the 10 files from the first group of random XML files generated from DTD-2 is taken and swapped from each other. This will form the test case 11 and test case 21. Essentially, test case 11 will contain 90 random XML files from DTD-1 and 10 random XML files from DTD-2; whereas test case 21 will contain 90 random XML files from DTD-2 and 10 random XML files from DTD-1. Similarly, we are repeating the same procedure for all other test cases by swapping 10 files from respective test cases of DTD-1 and DTD-2. As the result of this swapping operation, we designed the rest of the test cases containing majority of the XML files from DTD-1 and we name them test case 11, 12, 13, 14 and 15. The test cases containing majority of the XML files from DTD-2 are named as test case 21, 22, 23, 24 and 25. Each of the test cases are separately executed in our implementation of the technique that we were talking about in our paper and the output DTD obtained is noted. We are naming the output for test case 11 as DTD 11 and the name naming terminology has been followed for all other test cases and because of the use of noise XML files in to the original XML files, the original DTD to be generated is altered and a variety of DTD's are generated. By mapping these generated DTD's, we would be able to determine the optimum frequency to be set for our algorithm.

CHAPTER 5: CONCLUSION

Inferring a concise and accurate schema from a large collection of XML documents is important for XML data management. Based on Xtract, we proposed using NFA simulation in place of regular expression matching to improve the efficiency of XML schema inference. We have also proposed using a weighted target function to balance the generality and preciseness of the inferred schema, which can be important for noise reduction in XML classification/clustering.

The significant advantage in using our proposed XML schema extraction algorithm to the existing algorithm is that we can alter the amount of accuracy or the generalization that the schema needs. XML schema extraction algorithm is able to match the regular expressions by using NFA which simplified the processes of joining the individual candidate schemas and decreases the time taken for execution considerably. We are also exploring the possible level of complexity that the regular expression in practise contains, so that we can more easily customise frequency value for each user.

The proposed XML schema extraction algorithm can handle a wide range of heterogeneous XML documents collection. The efficiency of the schema generation process is increases many times by our proposed XML schema extraction algorithm. The quality of the generated schema is improved considerably by our proposed XML schema extraction algorithm. We investigated the usability of this algorithm and the potential user who may be any one ranging from a data miner in a complex situation to an ordinary user in a simpler situation. We introduced the ideas for implementing the algorithms efficiently with respect to time and memory constraints. Flexible and efficient methods of the practical XML schema generation step are explored.

APPENDIX

In chapter 4 the test cases generated from the synthesized DTDs DTD-1 and DTD-2 [23] are used in the experiment to map the optimum frequency value. DTD-1 and DTD-2 are used to generate 500 samples of XML files each by using the XML generator. The test cases are divided in to 100 files each by taking 90 files from DTD-1 and 10 files from DTD-2 and name them test case 11, test case 12, test case 13, test case 14 and test case 15. Similarly the other test cases are generated by taking 90 files from DTD-2 and 10 files from DTD-1 and name them test case 21, test case 22, test case 23, test case 24 and test case 25. The test cases are designed to make sure that we could efficiently map the frequency value from the generated DTD in our method. The DTD-1 and DTD-2 used in our experiments are shown here.

DTD-1

```
    <!-- FORMAT
************************************************************** -->

    <!-- RNAML: The root of any rnaml document.
**************************** -->
    <!ELEMENT rnaml (molecule)*>
    <!ATTLIST rnaml
          version (1.1)   #REQUIRED
       comment       CDATA   #IMPLIED
          reference-ids  IDREFS  #IMPLIED
       analysis-ids   IDREFS  #IMPLIED
          database-ids   IDREFS  #IMPLIED>


    <!-- MOLECULE: A given RNA molecule.
*********************************** -->
    <!ELEMENT molecule (identity?, sequence*, structure?)>
```

```
      <!ATTLIST molecule
            id            ID          #REQUIRED
          type          (rna | dna)  "rna"
          comment       CDATA       #IMPLIED
            reference-ids  IDREFS      #IMPLIED
          analysis-ids   IDREFS      #IMPLIED
            database-ids   IDREFS      #IMPLIED>


      <!-- IDENTITY: The symbolic description of the RNA molecule.          --
>
      <!ELEMENT identity (name, taxonomy?)>
      <!ATTLIST identity
            comment       CDATA   #IMPLIED
            reference-ids  IDREFS  #IMPLIED
          analysis-ids   IDREFS  #IMPLIED
            database-ids   IDREFS  #IMPLIED>

      <!ELEMENT name (#PCDATA)>                              <!-- type: string --
>

      <!-- SEQUENCE: A description of the rna sequence.                  -->
      <!ELEMENT sequence (numbering-system*, seq-data?, seq-annotation?)>
      <!ATTLIST sequence
          strand    CDATA         #IMPLIED
            length    CDATA         #IMPLIED
          circular  (true | false) #IMPLIED
            comment       CDATA    #IMPLIED
            reference-ids  IDREFS    #IMPLIED
          analysis-ids   IDREFS    #IMPLIED
            database-ids   IDREFS    #IMPLIED>


      <!-- NUMBERING SYSTEM: A numbering-system provides how the
bases are   -->
      <!-- numbered.  used-in-file indicates if the numbering-system is the  -->
      <!-- one used throughout the file.  If no numbering system is provided -->
      <!-- or used-in-file, the numbering from 1 to N is used.              -->
      <!ELEMENT numbering-system ((numbering-range)* | numbering-
table)>
      <!ATTLIST numbering-system
            id           ID     #REQUIRED
            used-in-file  (true | false) "false"
          comment       CDATA   #IMPLIED
            reference-ids  IDREFS  #IMPLIED
          analysis-ids   IDREFS  #IMPLIED
```

```
            database-ids   IDREFS  #IMPLIED>

      <!ELEMENT numbering-range (start, (end | length))>
      <!ATTLIST numbering-range
            comment        CDATA   #IMPLIED
            reference-ids  IDREFS  #IMPLIED
          analysis-ids   IDREFS  #IMPLIED
            database-ids   IDREFS  #IMPLIED>

      <!ELEMENT numbering-table (#PCDATA)> <!-- type: space delimited
integers -->
      <!ATTLIST numbering-table
            length    CDATA   #REQUIRED
          comment        CDATA   #IMPLIED
            reference-ids  IDREFS  #IMPLIED
          analysis-ids   IDREFS  #IMPLIED
            database-ids   IDREFS  #IMPLIED>


      <!-- SEQ-DATA: The actual data for a sequence.  The IUPAC (1984)       -
->
      <!-- symbols should be used:                              -->
      <!--         G=G, A=A, U=U, C=C, R=G|A, Y=U|C, M=A|C, K=G|U,
-->
      <!--         S=G|C, W=A|U, H=A|C|U, B=G|U|C,                        -->
      <!--         V=G|C|A, D=G|A|U, N=G|A|U|C                          -->

      <!ELEMENT seq-data (#PCDATA)>                    <!-- type: string -->
      <!ATTLIST seq-data
          comment        CDATA   #IMPLIED
            reference-ids  IDREFS  #IMPLIED
          analysis-ids   IDREFS  #IMPLIED
            database-ids   IDREFS  #IMPLIED>


      <!-- SEQ-ANNOTATION                                          -->
      <!ELEMENT seq-annotation (modification | segment)*>
      <!ATTLIST seq-annotation
            comment        CDATA   #IMPLIED
            reference-ids  IDREFS  #IMPLIED
          analysis-ids   IDREFS  #IMPLIED
            database-ids   IDREFS  #IMPLIED>

      <!ELEMENT modification  (base-id, modified-type)>
      <!ATTLIST modification
            comment        CDATA   #IMPLIED
            reference-ids  IDREFS  #IMPLIED
          analysis-ids   IDREFS  #IMPLIED
            database-ids   IDREFS  #IMPLIED>

      <!ELEMENT modified-type (#PCDATA)>                 <!-- type: string -->

      <!ELEMENT segment  (seg-name?, base-id-5p, (base-id-3p | length))>
      <!ATTLIST segment
            comment       CDATA   #IMPLIED
            reference-ids  IDREFS  #IMPLIED
          analysis-ids   IDREFS  #IMPLIED
            database-ids   IDREFS  #IMPLIED>

      <!ELEMENT seg-name (#PCDATA)>                    <!-- type: string -->

      <!-- STRUCTURE: A description of the structure of the rna.           -->
      <!-- name: A description of the structure.                          -->
      <!ELEMENT structure (model)*>
      <!ATTLIST structure
            comment       CDATA   #IMPLIED
            reference-ids  IDREFS  #IMPLIED
          analysis-ids   IDREFS  #IMPLIED
            database-ids   IDREFS  #IMPLIED>

      <!-- MODEL: A model for the RNA structure.                         -->
      <!ELEMENT model (model-info?, base*, str-annotation?, secondary-
  structure-display?)>
      <!ATTLIST model
            id  ID  #IMPLIED
          comment       CDATA   #IMPLIED
            reference-ids  IDREFS  #IMPLIED
          analysis-ids   IDREFS  #IMPLIED
            database-ids   IDREFS  #IMPLIED>

      <!ELEMENT model-info (method?, resolution?, free-energy*)>
      <!ATTLIST model-info
            comment       CDATA   #IMPLIED
            reference-ids  IDREFS  #IMPLIED
          analysis-ids   IDREFS  #IMPLIED
            database-ids   IDREFS  #IMPLIED>

      <!ELEMENT method      (#PCDATA)>                 <!-- type: string -->
      <!ATTLIST method
            comment       CDATA   #IMPLIED
            reference-ids  IDREFS  #IMPLIED
          analysis-ids   IDREFS  #IMPLIED
```

```
           database-ids   IDREFS  #IMPLIED>

      <!ELEMENT resolution  (#PCDATA)>     <!-- type: float, unit: angstroms
-->
      <!ATTLIST resolution
           comment       CDATA   #IMPLIED
           reference-ids  IDREFS  #IMPLIED
         analysis-ids   IDREFS  #IMPLIED
           database-ids   IDREFS  #IMPLIED>

      <!ELEMENT free-energy (#PCDATA)>     <!-- type: float, unit:
kcal/mole -->
      <!ATTLIST free-energy
           comment       CDATA   #IMPLIED
           reference-ids  IDREFS  #IMPLIED
         analysis-ids   IDREFS  #IMPLIED
           database-ids   IDREFS  #IMPLIED>

      <!-- BASE: A base is a residue in secondary or tertiary structure.     -->
      <!ELEMENT base      (strand?, position, base-type?, alt-loc?, insertion?,
atom*)>
      <!ATTLIST base
           comment       CDATA   #IMPLIED
           reference-ids  IDREFS  #IMPLIED
         analysis-ids   IDREFS  #IMPLIED
           database-ids   IDREFS  #IMPLIED>

      <!ELEMENT base-type (#PCDATA)>                   <!-- type: string -->
      <!ELEMENT alt-loc   (#PCDATA)>                  <!-- type: char   -->
      <!ELEMENT insertion (#PCDATA)>                   <!-- type: char   -->


      <!-- ATOM: An atom with support for PDB fields.                    -->
      <!ELEMENT atom (atom-type, coordinates?, occupancy?,
                temp-factor?, seg-id?, element?, charge?)>
      <!ATTLIST atom
           serial        CDATA   #IMPLIED
         comment       CDATA   #IMPLIED
           reference-ids  IDREFS  #IMPLIED
         analysis-ids   IDREFS  #IMPLIED
           database-ids   IDREFS  #IMPLIED>

      <!ELEMENT atom-type   (#PCDATA)>                 <!-- type: string -->
      <!ELEMENT coordinates (#PCDATA)>     <!-- three space delimited
floats -->
      <!ELEMENT occupancy   (#PCDATA)>                 <!-- type: float  -->
      <!ELEMENT temp-factor (#PCDATA)>                  <!-- type: float  -->
      <!ELEMENT seg-id      (#PCDATA)>                <!-- type: string -->
      <!ELEMENT element     (#PCDATA)>                <!-- type: string -->
      <!ELEMENT charge      (#PCDATA)>                <!-- type: string -->


      <!-- STR-ANNOTATION:                                        -->
      <!ELEMENT str-annotation ((base-conformation | base-pair |
                         base-triple | base-stack | helix |
                            pseudoknot |
                            single-strand | distance-constraint |
                            surface-constraint)*)>
      <!ATTLIST str-annotation
            comment       CDATA   #IMPLIED
            reference-ids  IDREFS  #IMPLIED
          analysis-ids   IDREFS  #IMPLIED
            database-ids   IDREFS  #IMPLIED>


      <!-- BASE-CONFORMATION:                                        -->
      <!ELEMENT base-conformation (base-id, pucker?, glycosyl?,
                          base-torsion-angles?)>
      <!ATTLIST base-conformation
            comment       CDATA   #IMPLIED
            reference-ids  IDREFS  #IMPLIED
          analysis-ids   IDREFS  #IMPLIED
            database-ids   IDREFS  #IMPLIED>

      <!ELEMENT pucker   (#PCDATA)>                    <!-- type: string -->
      <!ELEMENT glycosyl (#PCDATA)>                          <!-- type: string
-->

      <!ELEMENT base-torsion-angles (alpha?, beta?, gamma?, delta?,
epsilon?,
                           zeta?, chi?, nu0?, nu1?, nu2?, nu3?, nu4?)>
      <!ATTLIST base-torsion-angles
            comment       CDATA   #IMPLIED
            reference-ids  IDREFS  #IMPLIED
          analysis-ids   IDREFS  #IMPLIED
            database-ids   IDREFS  #IMPLIED>

      <!ELEMENT alpha   (#PCDATA)>          <!-- type: float, unit: degrees --
>
      <!ELEMENT beta    (#PCDATA)>         <!-- type: float, unit: degrees -->
      <!ELEMENT gamma   (#PCDATA)>          <!-- type: float, unit: degrees
-->
```

```
    <!ELEMENT delta   (#PCDATA)>          <!-- type: float, unit: degrees -->
    <!ELEMENT epsilon (#PCDATA)>           <!-- type: float, unit: degrees --
>
    <!ELEMENT zeta    (#PCDATA)>          <!-- type: float, unit: degrees -->
    <!ELEMENT chi     (#PCDATA)>          <!-- type: float, unit: degrees -->
    <!ELEMENT nu0     (#PCDATA)>          <!-- type: float, unit: degrees --
>
    <!ELEMENT nu1     (#PCDATA)>          <!-- type: float, unit: degrees --
>
    <!ELEMENT nu2     (#PCDATA)>          <!-- type: float, unit: degrees --
>
    <!ELEMENT nu3     (#PCDATA)>          <!-- type: float, unit: degrees --
>
    <!ELEMENT nu4     (#PCDATA)>          <!-- type: float, unit: degrees --
>



    <!-- BASE-PAIR: A base pair requires two bases from the same molecule
-->
    <!-- (molecule id unnecessary) or from two molecules.                -->
    <!ELEMENT base-pair (base-id-5p, base-id-3p, edge-5p?, edge-3p?,
                 bond-orientation?, strand-orientation?)>
    <!ATTLIST base-pair
        comment        CDATA   #IMPLIED
          reference-ids  IDREFS  #IMPLIED
        analysis-ids   IDREFS  #IMPLIED
          database-ids   IDREFS  #IMPLIED>
    <!ELEMENT base-id-5p        (base-id)>            <!-- type: string -->
    <!ELEMENT base-id-3p        (base-id)>            <!-- type: string -->
    <!ELEMENT edge-5p           (#PCDATA)>            <!-- type: string -->
    <!ELEMENT edge-3p           (#PCDATA)>            <!-- type: string -->
    <!ELEMENT bond-orientation   (#PCDATA)>            <!-- type: string --
>
    <!ELEMENT strand-orientation (#PCDATA)>            <!-- type: string --
>



    <!-- BASE-TRIPLE: A base triple is formed of two or three base-pairs.  --
>
    <!ELEMENT base-triple ((base-pair | base-pair-id),
                  (base-pair | base-pair-id),
                  (base-pair | base-pair-id))>
    <!ATTLIST base-triple
        comment        CDATA   #IMPLIED
          reference-ids  IDREFS  #IMPLIED
        analysis-ids   IDREFS  #IMPLIED
        database-ids   IDREFS  #IMPLIED>

<!-- BASE-STACK:                                   -->
<!ELEMENT base-stack (base-id, base-id)>
<!ATTLIST base-stack
     comment       CDATA   #IMPLIED
        reference-ids  IDREFS  #IMPLIED
     analysis-ids   IDREFS  #IMPLIED
        database-ids   IDREFS  #IMPLIED>

<!-- HELIX:                                    -->
<!ELEMENT helix (base-id-5p, base-id-3p, length)>
<!ATTLIST helix
        id ID  #IMPLIED
     comment       CDATA   #IMPLIED
        reference-ids  IDREFS  #IMPLIED
     analysis-ids   IDREFS  #IMPLIED
        database-ids   IDREFS  #IMPLIED>

<!-- PSEUDOKNOT:                                   -->
<!ELEMENT pseudoknot (helix-id, helix-id)>
<!ATTLIST pseudoknot
     comment       CDATA   #IMPLIED
        reference-ids  IDREFS  #IMPLIED
     analysis-ids   IDREFS  #IMPLIED
        database-ids   IDREFS  #IMPLIED>

<!ELEMENT helix-id EMPTY>
<!ATTLIST helix-id
        ref          IDREF   #REQUIRED
     comment       CDATA   #IMPLIED
        reference-ids  IDREFS  #IMPLIED
     analysis-ids   IDREFS  #IMPLIED
        database-ids   IDREFS  #IMPLIED>


<!-- SINGLE-STRAND: A single strand is a segment!                 -->
<!ELEMENT single-strand (segment)>
<!ATTLIST single-strand
        comment       CDATA   #IMPLIED
        reference-ids  IDREFS  #IMPLIED
     analysis-ids   IDREFS  #IMPLIED
        database-ids   IDREFS  #IMPLIED>


<!-- DISTANCE-CONSTRAINT:                               -->
```

```
    <!ELEMENT distance-constraint (base-id, atom-type,
                             base-id, atom-type,
                               mean, range?, weight?)>
    <!ATTLIST distance-constraint
          comment       CDATA   #IMPLIED
          reference-ids  IDREFS  #IMPLIED
        analysis-ids   IDREFS  #IMPLIED
          database-ids   IDREFS  #IMPLIED>
    <!ELEMENT mean   (#PCDATA)>         <!-- type: float, unit: angstroms
-->
    <!ELEMENT range  (#PCDATA)>         <!-- type: float, unit: angstroms -
->
    <!ELEMENT weight (#PCDATA)>                        <!-- type: float -->


    <!-- SURFACE-CONSTRAINT:                                    -->
    <!ELEMENT surface-constraint (base-id, atom-type, surface-value?)>
    <!ATTLIST surface-constraint
          comment       CDATA   #IMPLIED
          reference-ids  IDREFS  #IMPLIED
        analysis-ids   IDREFS  #IMPLIED
          database-ids   IDREFS  #IMPLIED>
    <!ELEMENT surface-value (#PCDATA)>                <!-- type: float -->


    <!-- SECONDARY-STRUCTURE-DISPLAY                                  -->
    <!-- X,Y coordinates can be defined for the sequence and then         -->
    <!-- used to display the secondary structure.                        -->
    <!ELEMENT secondary-structure-display (ss-base-coord)*>
    <!ATTLIST secondary-structure-display comment        CDATA
#IMPLIED
          reference-ids  IDREFS  #IMPLIED
        analysis-ids   IDREFS  #IMPLIED
          database-ids   IDREFS  #IMPLIED>

    <!-- This provides an x,y position for a base.                -->
    <!ELEMENT ss-base-coord (base-id, coordinates)>
    <!ATTLIST ss-base-coord
          comment       CDATA   #IMPLIED
          reference-ids  IDREFS  #IMPLIED
        analysis-ids   IDREFS  #IMPLIED
          database-ids   IDREFS  #IMPLIED>

    <!-- GENERAL ELEMENTS
************************************************ -->
```

```
    <!ELEMENT url  (#PCDATA)>                       <!-- type: string -->
    <!ELEMENT file (#PCDATA)>                       <!-- type: string -->

    <!ELEMENT path (url | file)>

    <!ELEMENT person (first-name, last-name, affiliation*)>
    <!ELEMENT first-name   (#PCDATA)>                 <!-- type: string -->
    <!ELEMENT last-name    (#PCDATA)>                      <!-- type: string
-->
    <!ELEMENT affiliation (#PCDATA)>                  <!-- type: string -->

    <!ELEMENT program (prog-name | prog-version | reference)*>
    <!ELEMENT prog-name    (#PCDATA)>                 <!-- type: string -->
    <!ELEMENT prog-version (#PCDATA)>                      <!-- type: string
-->

    <!ELEMENT date (day?, month?, year)>
    <!ELEMENT day   (#PCDATA)>                      <!-- type: integer -->
    <!ELEMENT month (#PCDATA)>                           <!-- type: integer
-->
    <!ELEMENT year  (#PCDATA)>                           <!-- type: integer
-->

    <!ELEMENT position (#PCDATA)>                        <!-- type: integer
-->
    <!ELEMENT start  (#PCDATA)>                     <!-- type: integer -->
    <!ELEMENT end    (#PCDATA)>                          <!-- type: integer
-->
    <!ELEMENT length (#PCDATA)>                          <!-- type: integer
-->

    <!ELEMENT base-id (molecule-id?, model-id?, strand?, position)>

    <!ELEMENT molecule-id EMPTY>
    <!ATTLIST molecule-id
          ref  IDREF  #REQUIRED>

    <!ELEMENT model-id EMPTY>
    <!ATTLIST model-id
          ref  IDREF  #IMPLIED>

    <!ELEMENT strand (#PCDATA)>                          <!-- type: string
-->

    <!ELEMENT base-pair-id EMPTY>
    <!ATTLIST base-pair-id
```

```
            ref  IDREF  #REQUIRED>

      <!ELEMENT alignment-id EMPTY>
      <!ATTLIST alignment-id
            ref  IDREF  #REQUIRED>
```

DTD-2

```
      <!—FORMAT
*********************************************************** -->

      <!-- RNAML: The root of any rnaml document.
*************************** -->
      <!ELEMENT rnaml (molecule | database-entry | analysis | revision)*>
      <!ATTLIST rnaml
            version (1.1)   #REQUIRED
          comment       CDATA   #IMPLIED
            reference-ids  IDREFS  #IMPLIED
          analysis-ids   IDREFS  #IMPLIED
            database-ids   IDREFS  #IMPLIED>


      <!-- MOLECULE: A given RNA molecule.
********************************** -->
      <!ELEMENT molecule (identity?, sequence*, structure?)>
      <!ATTLIST molecule
            id            ID          #REQUIRED
          type          (rna | dna)  "rna"
          comment       CDATA       #IMPLIED
            reference-ids  IDREFS      #IMPLIED
          analysis-ids   IDREFS      #IMPLIED
            database-ids   IDREFS      #IMPLIED>


      <!-- IDENTITY: The symbolic description of the RNA molecule.          --
>
      <!ELEMENT identity (name, taxonomy?)>
      <!ATTLIST identity
            comment       CDATA   #IMPLIED
            reference-ids  IDREFS  #IMPLIED
          analysis-ids   IDREFS  #IMPLIED
```

```
           database-ids   IDREFS  #IMPLIED>

      <!ELEMENT name (#PCDATA)>                           <!-- type: string --
>

      <!-- SEQUENCE: A description of the rna sequence.                    -->
      <!ELEMENT sequence (numbering-system*, seq-data?, seq-annotation?)>
      <!ATTLIST sequence
         strand    CDATA        #IMPLIED
           length    CDATA         #IMPLIED
         circular  (true | false) #IMPLIED
           comment       CDATA     #IMPLIED
           reference-ids  IDREFS    #IMPLIED
         analysis-ids   IDREFS    #IMPLIED
           database-ids   IDREFS    #IMPLIED>



      <!-- NUMBERING SYSTEM: A numbering-system provides how the
bases are   -->
      <!-- numbered.  used-in-file indicates if the numbering-system is the  -->
      <!-- one used throughout the file.  If no numbering system is provided -->
      <!-- or used-in-file, the numbering from 1 to N is used.              -->
      <!ELEMENT numbering-system ((numbering-range)* | numbering-
table)>
      <!ATTLIST numbering-system
           id           ID     #REQUIRED
           used-in-file  (true | false) "false"
         comment       CDATA   #IMPLIED
           reference-ids  IDREFS  #IMPLIED
         analysis-ids   IDREFS  #IMPLIED
           database-ids   IDREFS  #IMPLIED>

      <!ELEMENT numbering-range (start, (end | length))>
      <!ATTLIST numbering-range
           comment       CDATA   #IMPLIED
           reference-ids  IDREFS  #IMPLIED
         analysis-ids   IDREFS  #IMPLIED
           database-ids   IDREFS  #IMPLIED>

      <!ELEMENT numbering-table (#PCDATA)> <!-- type: space delimited
integers -->
      <!ATTLIST numbering-table
           length    CDATA   #REQUIRED
         comment       CDATA   #IMPLIED
           reference-ids  IDREFS  #IMPLIED
         analysis-ids   IDREFS  #IMPLIED
```

```
          database-ids   IDREFS  #IMPLIED>



     <!-- SEQ-DATA: The actual data for a sequence.  The IUPAC (1984)       -
->
     <!-- symbols should be used:                                   -->
     <!--         G=G, A=A, U=U, C=C, R=G|A, Y=U|C, M=A|C, K=G|U,
-->
     <!--         S=G|C, W=A|U, H=A|C|U, B=G|U|C,                        -->
     <!--         V=G|C|A, D=G|A|U, N=G|A|U|C                           -->

     <!ELEMENT seq-data (#PCDATA)>                   <!-- type: string -->
     <!ATTLIST seq-data
        comment       CDATA   #IMPLIED
          reference-ids  IDREFS  #IMPLIED
        analysis-ids   IDREFS  #IMPLIED
          database-ids   IDREFS  #IMPLIED>



     <!-- SEQ-ANNOTATION                                              -->
     <!ELEMENT seq-annotation (modification | segment)*>
     <!ATTLIST seq-annotation
          comment       CDATA   #IMPLIED
          reference-ids  IDREFS  #IMPLIED
        analysis-ids   IDREFS  #IMPLIED
          database-ids   IDREFS  #IMPLIED>

     <!ELEMENT modification  (base-id, modified-type)>
     <!ATTLIST modification
          comment       CDATA   #IMPLIED
          reference-ids  IDREFS  #IMPLIED
        analysis-ids   IDREFS  #IMPLIED
          database-ids   IDREFS  #IMPLIED>

     <!ELEMENT modified-type (#PCDATA)>               <!-- type: string -->

     <!ELEMENT segment  (seg-name?, base-id-5p, (base-id-3p | length))>
     <!ATTLIST segment
          comment       CDATA   #IMPLIED
          reference-ids  IDREFS  #IMPLIED
        analysis-ids   IDREFS  #IMPLIED
          database-ids   IDREFS  #IMPLIED>

     <!ELEMENT seg-name (#PCDATA)>                    <!-- type: string -->
     <!-- DATABASE-ENTRY:                                              -->
     <!ELEMENT database-entry (database, entry, path?)>
```

```
    <!ATTLIST database-entry
          id       ID     #REQUIRED
       comment  CDATA  #IMPLIED
       reference-ids  IDREFS  #IMPLIED
       analysis-ids   IDREFS  #IMPLIED
          database-ids   IDREFS  #IMPLIED>
    <!ELEMENT database (#PCDATA)>                   <!-- type: string -->
    <!ELEMENT entry    (#PCDATA)>                  <!-- type: string -->


    <!-- ANALYSIS:                                  -->
    <!ELEMENT analysis (program | date | author | reference)*>
    <!ATTLIST analysis
          id      ID    #REQUIRED
       comment       CDATA   #IMPLIED
       reference-ids  IDREFS  #IMPLIED
       analysis-ids   IDREFS  #IMPLIED
          database-ids   IDREFS  #IMPLIED>


    <!-- REVISION:                                  -->
    <!ELEMENT revision (#PCDATA | date)*>
    <!ATTLIST revision
          comment       CDATA   #IMPLIED
          reference-ids  IDREFS  #IMPLIED
       analysis-ids   IDREFS  #IMPLIED
          database-ids   IDREFS  #IMPLIED>


    <!-- GENERAL ELEMENTS
********************************************** -->

    <!ELEMENT url  (#PCDATA)>                     <!-- type: string -->
    <!ELEMENT file (#PCDATA)>                     <!-- type: string -->

    <!ELEMENT path (url | file)>

    <!ELEMENT person (first-name, last-name, affiliation*)>
    <!ELEMENT first-name   (#PCDATA)>               <!-- type: string -->
    <!ELEMENT last-name    (#PCDATA)>                    <!-- type: string
-->
    <!ELEMENT affiliation (#PCDATA)>               <!-- type: string -->

    <!ELEMENT program (prog-name | prog-version | reference)*>
    <!ELEMENT prog-name    (#PCDATA)>               <!-- type: string -->
    <!ELEMENT prog-version (#PCDATA)>                    <!-- type: string
```

```
-->

     <!ELEMENT date (day?, month?, year)>
     <!ELEMENT day   (#PCDATA)>                        <!-- type: integer -->
     <!ELEMENT month (#PCDATA)>                                <!-- type: integer
-->
     <!ELEMENT year  (#PCDATA)>                                <!-- type: integer
-->

     <!ELEMENT position (#PCDATA)>                             <!-- type: integer
-->
     <!ELEMENT start  (#PCDATA)>                       <!-- type: integer -->
     <!ELEMENT end    (#PCDATA)>                               <!-- type: integer
-->
     <!ELEMENT length (#PCDATA)>                               <!-- type: integer
-->

     <!ELEMENT base-id (molecule-id?, model-id?, strand?, position)>

     <!ELEMENT molecule-id EMPTY>
     <!ATTLIST molecule-id
            ref  IDREF  #REQUIRED>

     <!ELEMENT model-id EMPTY>
     <!ATTLIST model-id
            ref  IDREF  #IMPLIED>

     <!ELEMENT strand (#PCDATA)>                                <!-- type: string
-->

     <!ELEMENT base-pair-id EMPTY>
     <!ATTLIST base-pair-id
            ref  IDREF  #REQUIRED>

     <!ELEMENT alignment-id EMPTY>
     <!ATTLIST alignment-id
            ref  IDREF  #REQUIRED>
```

BIBILIOGRAPHY

1. Abiteboul, S (1997). Querying semi-structured data. In F.N. Afrati and P. G. Kolaitis (Eds.), Proceedings of the ICDT Conference (pp. 1-18). Berlin Heidelberg New York: Springer.

2. Bertino, E., Guerrini, G., & Mesiti, M. (2004). A matching algorithm for measuring the structural similarity between an XML document and a DTD and its applications. Information Systems, 29(1), 23-46.

3. Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., & Yergeau, F. (2004). Extensible Markup Language (XML) 1.0 (Third Edition). W3C, from http://www.w3.org/TR/2004/REC-xml-20040204/.

4. Canfield, R., Xing, G. (2005). Approximate matching of XML document with regular hedge grammar. International Journal of Computer Mathematics. 82(10), 1191-1198.

5. Denoyer, L., Gallinari, P., & Vercoustre A.(2006). Report on the XML Mining Track at INEX 2005 and INEX 2006. In Fuhr, N., Lalmas, M., &Trotman, A. (Ed.) INEX 2006 Lecture Notes in Computer Science (4518) (pp. 432-443). Berlin Heidelberg New York: Springer.

6. Garofalakis, M. N., Gionis, A., Rastogi, R., Seshadri, S., & Shim, K. (2000). XTRACT: A system for extracting document type descriptors from XML documents. In W. Chen, J.F. Naughton, and P.A. Bernstein (Eds.), Proceedings of the International Conference on Management of Data (pp. 165–176). New York: ACM.

7. Ugo Galassi and Attilio Giordana, 2005. Learning regular expressions from noisy sequences. In SARA, 92-106.

8. Hopcroft, J.E., Motwani, R., & Ullman, J.D. (2006). Introduction to Automata Theory, Languages, and Computation, 3rd Edition, Addison-Wesley.

9. Jagadish, H.V., Al-Khalifa, S., Chapman, A., Lakshmanan, L., Nierman, A., Paparizos, S., Patel, J.M., Srivastava, D., Wiwatwattana, N., Wu, Y., & Yu, C. (2002). Timber: A native XML database. VLDB Journal. 11(4), 274–291.

10. Yunyao Li, Rajasekar Krishnamurthy, Sriram Raghavan, Shivakumar Vaithyanathan, and H. V. Jagadish. (2008). Regular expression learning for information extraction. In Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP '08). Association for Computational Linguistics, Morristown, NJ, USA, 21-30.

11. Xing, G.(2006). Fast Approximate Matching Between XML Documents and Schemata. In X. Zhou, J. Li, H. Shen, M. Kitsuregawa and Y. Zhang(Eds.), APWeb 2006 Lecture Notes in Computer Science (3841) (pp. 425-436). Berlin Heidelberg New York: Springer.

12. Xing. G, Guo. J, and Xia. Z (2006). Classifying XML Documents Based on Structure/Content Similarity. In N. Fuhr, M. Lalmas and A. Trotman (Eds.), INEX 2006 Lecture Notes in Computer Science (4518) (pp. 444-457). Berlin Heidelberg New York: Springer.

13. XML Generator can be retrieved from http://www.alphaworks.ibm.com/tech/xmlgenerator.

14. Jan Hegewald, Felix Naumann, Melanie Weis. XStruct: Efficient Schema Extraction from Multiple and Large XML Documents, icdew, pp.81, 22nd International Conference on Data Engineering Workshops (ICDEW'06), 2006.

15. J.-K. Min, J.-Y. Ahn, and C.-W. Chung. Efficient ex-traction of schemas for XML documents. In Information Processing Letters, 2003.

16. J. Rissanen, Modeling by shortest data description. Automatica, 14:465–471, 1978.

17. Lili Sun and Yan Li, XML Schema. XML Documents with Usage Control. In IJCSNS International Journal of Computer Science and Network 170 Security, VOL.7 No.10, October 2007.

18. Makoto Murata from IBM Tokyo Research Lab, Dongwon Lee from Penn State University Murali Mani from Worcester Polytechnic Institute and Kohsuke Kawaguchi from Sun Microsystems. Taxonomy of XML Schema Languages using Formal Language Theory. In Extreme Mark-up Languages 2000.

19. Mohamad Alishahi, Mahmoud Naghibzadeh and Baharak Shakeri Aski. Tag-name based clustering of XML documents. In International Journal of Computer and Electrical Engineering, Vol. 2, No. 1, February, 2010 1793-8163.

20. Richi Nayak from Queensland University of Technology, School of Information Systems, Brisbane, Australia. Fast and effective clustering of XML data using structural information. In Knowledge and Information Systems, Volume 14, Issue 2, January 2008.

21. R. Nayak. XML Data Mining: Process and Applications. In Song, Min and Wu, Yi-Fang, Eds, Idea Group Inc. /IGI Global, 2008.

22. The Wisconsin's XML data bank. Accessed from: http://www.cs.wisc.edu/hiagara/data.html.

23. The XML data repository. Accessed from:

http://www.cs.washington.edu/research/xmldatasets/.

24. Michael Ley. Book DTD. Accessed from: http://www.informatik.uni-trier.de/~ley/db/about/dblp.dtd.

25. P. Antonellis, C. Makris, N. Tsirakis. XEdge: clustering homogeneous and heterogeneous XML documents using edge summaries. In SAC 2008, pp. 1081-1088, 2008.

26. Supoj Sutanthavibul, Brian V.Smith and Paul King. An interactive drawing tool named Xfig. Accessed from: http://epb.lbl.gov/xfig/frm_introduction.html.