XBeGene: Scalable XML Documents Generator By Example Based on Real Data

Manami Harazaki*, Joe Tekli‡[1], Shohei Yokoyama†, Naoki Fukuta†, Richard Chbeir‡ and Hiroshi Ishikawa†
*Graduate School of Informatics,
Shizuoka University, Hamamatsu-shi, Shizuoka, Japan 432–8011
Email: gs09046@s.inf.shizuoka.ac.jp
†Department of Computer Science, Faculty of Informatics,
Shizuoka University, Hamamatsu-shi, Shizuoka, Japan 432–8011
Email: yokoyama, fukuta, ishikawa@inf.shizuoka.ac.jp
‡LE2I Laboratory CNRS, University of Bourgogne, 21076 Dijon, France
Email: joe.tekli, richard.chbeir@u-bourgogne.fr

***Abstract*—XML datasets of various sizes and properties are needed to evaluate the correctness and efficiency of XML-based algorithms and applications. While several downloadable datasets can be found online, these are predefined by system experts and might not be suitable to evaluate every algorithm. Tools for generating synthetic XML documents underline an alternative solution, promoting flexibility and adaptability in generating synthetic document collections. Nonetheless, the usefulness of existing XML generators remains rather limited due to the restricted levels of expressiveness allowed to users (limited control over XML structural and content properties). In this paper, we develop a novel XML By example Generator (XBeGene) for producing synthetic XML data which closely reflect the user's requirements. Inspired by the query-by-example paradigm in information retrieval, our generator system i) allows the user to provide her own sample XML documents as input, ii) analyzes the structure, occurrence frequencies, and content distributions for each XML element in the user input documents, and iii) produces synthetic XML documents which closely concur, in both structural and content features, to the user's input data. The size of each synthetic document as well as that of the entire document collection are also specified by the user. In addition, our approach is designed to efficiently generate large collections of documents, and runs in average polynomial time. Clustering experiments demonstrate high correlation levels between the specified user requirements and the characteristics of the generated XML data, while timing results confirm our approach's scalability to large scale document collections.**

***Keywords*-XML; data generator; benchmark;**

I. INTRODUCTION

Artificial collections of XML documents have many applications in benchmarking, testing, and evaluation of several algorithms, tools, and systems. For instance, methods for XML similarity evaluation [1], [2], [3] , version control and change management [4], [5], document classification [6], [7] and clustering [8], [9], all require large XML test collections in order to evaluate their effectiveness and/or efficiency levels. Nonetheless, even though the need is overwhelming, it is often difficult to find or generate appropriate XML data for testing. This is a significant stumbling block in system development since considerable resources and manual labor are required to generate test data. Furthermore, different applications require different kinds of XML documents with different characteristics. For instance, a benchmark for data-intensive applications might require large and homogeneous documents with many references among elements, whereas an adequate test suite for a parser might be an heterogeneous collection with thousands of documents of various sizes.

On one hand, several downloadable XML datasets can be found on the web. Such datasets may be real documents of various types, e.g., *Plays of Shakespear* [10], SIGMOD Record [11], DBLP [12], or XML benchmarks, e.g., [13], [14], [15]. Nonetheless, these are not always suitable to evaluate each and every application or algorithm. Most importantly, they are predefined by domain experts, and thus are not adaptable to each user's requirements and/or application constraints. On the other hand, a few tools to automatically generate XML data have been recently developed [16], [17], [18]. An XML generator is a program that produces synthetic XML documents according to given user constraints. While they seem more flexible than predefined XML datasets, the usefulness of existing generators remains rather limited due to the restricted levels of expressiveness allowed to users. XMLGen [16] produces XML documents with random structure and contents; ToXGene [17] focuses on content distributions, and provides limited control over the structure of the data; whereas GxBE [18] only generates XML document structures, disregarding contents.

In this paper, we propose an XML By example Generator (XBeGene) capable of producing synthetic XML data which closely reflect the structural characteristics and content distributions required by the user. Inspired by the *query-by-example* paradigm in information retrieval, our system: i) allows the user to provide her own sample XML documents as input, ii) analyzes the structure (i.e., parent/child relations, paths, hierarchical levels and ordering), occurrence frequencies (occurrence probability of each tag name), and content distributions (allowed values and their frequencies)

[1]The author is supported in part by the Japan Society for the Promotion of Science, JSPS Post-doc Fellowship no: PE10006.

for each XML element/attribute in the user input documents, and iii) produces synthetic XML documents which closely concur, in both structural and content features, to the user input data. The size of each synthetic document, as well as the size of the entire document collection (total number of documents) are also specified by the user. In other words, our method provides the user with full control over the different aspects of the XML data generation process. In addition, our approach is designed to efficiently generate large collections of documents, and runs in average polynomial time.

A battery of experiments was undertaken to test XBe-Gene, evaluating i) the generator's accuracy, estimating the similarity between real and synthetic XML documents using a document clustering method and ii) element probability correlation, comparing real and synthetic element/attribute occurrences. Experimental results confirm our algorithm's effectiveness and efficiency in generating synthetic XML documents of arbitrary sizes, in comparison with existing solutions, while preserving the structural and content features of user data.

II. Background

Since the last decade, XML has been used extensively in various application domains, ranging over databases (information interchange) [19], information retrieval (search and indexing of semi-structured data) [20], as well as for complex objects description, protein and gene encoding [22] , etc). Increasing demands in interoperability and flexibility, especially on the web, have underlined an overwhelming need for XML-based tools and applications. Nonetheless, XML application development requires proper testing and evaluation, which (in turn) emphasizes the need for large collections of relevant XML test data.

Although XML-based applications are becoming widespread, it is frequently difficult to find or generate appropriate input data for testing. This is a significant stumbling block in system development since considerable resources are required to generate test data. Furthermore, different applications require different kinds of XML documents with different characteristics. For instance, a benchmark for data-intensive applications might require large and homogeneous documents with many references among elements, whereas an adequate test suite for a parser might be an heterogeneous collection with thousands of documents of various sizes.

Currently, two distinct sources for XML documents are used for testing. First, several downloadable XML datasets can be found on the web. Such datasets might be real documents of various types, e.g., *Plays of Shakespear* [10], SIGMOD Record [11], DBLP [12], or XML benchmarks, e.g., XMark [13], XBench [14], Michigan Benchmark [15], TPoX [23], etc. Nonetheless, downloadable XML datasets are not always suitable to evaluate each and every application or algorithm. They are predefined by domain experts, and thus are not adaptable to each user's requirements and/or application constraints. Data generators are a second source of test data collections. While several data generators for relational databases have been developed, e.g. [13], [14], [15], [23], few XML document generators have been proposed to date [16], [17], [18]. An XML generator is a program that produces XML documents according to given user constraints. In fact, XML generators provide the user with flexibility in defining their XML documents of choice. However, their usefulness is determined by the level of expressiveness allowed to users in describing the properties desired in target documents. Following [18], an XML dataset should satisfy three main properties to be useful for effective system testing and evaluation:

(1) The data structure must conform to the structure of the target application.
(2) The datasets should correspond to the expected workload and have expected characteristics. For instance, the user should be able to specify whether an optional element is to appear frequently, or less often, in the synthetic document, or whether its branching factor is large.
(3) The data values must match the expected data distribution.

XMLGen [16] produces XML documents with random structures and contents. The user may only specify parameters that control the randomness of the result, e.g., the number of levels in the resulting tree, and the minimum and maximum number of children for a given level. In short, it does not seem to comply with any of the properties mentioned above.

ToXGene [17] focuses on content distributions (e.g., allowed element/attribute values and their frequencies), such as constraints are specified locally, within an XML schema. Nonetheless, despite ToXGene's rich definition of the XML content that should be produced, the ability to control the generated structure of the XML data is very limited. Since all constraints are specified locally in the schema, global structural properties of the document (e.g., a bound on the total document size, element/attribute occurrence frequencies, etc.) cannot be provided. Hence, ToXGen can generate datasets verifying properties 1 and 3 (but not 2).

In [18], the authors introduce GxBE to generate XML document structures that satisfy a given DTD grammar and a sample input XML document. It uses a natural declarative syntax, which includes XPath expressions to allow users to express global and local characteristics for the desired target documents (specification of types describing literals, elements, attributes, etc.), such as the generated documents can require satisfaction of XPath expressions from a given workload. While GxBE is clearly more sophisticated than its predecessors in considering the structural properties of XML data, it only targets XML structure and does not deal with

Table I
COMPARISON OF THE EXISTING APPROACHES

	Property1 Structure	Property2 Element/ Attribute characteristics	Property3 Element/ Attribute value distribution	Property4 XML Grammer
XMLGen [13]	x	x	x	Not required
ToXGene [17]	✓	x	✓	Required
GxBE [18]	✓	✓	x	Required
XBeGene (Our App.)	✓	✓	✓	Not required

XML content (i.e., element/attribute values are disregarded). In other words, the datasets generated using GxBE verify properties 1 and 2 (but not 3) (cf. Table I).

Note that the ToXGene [17] and GxBE [18] generators require XML grammars (DTD or XML schema) as input to the data generation task, so as to produce synthetic documents. Nonetheless, we assume that input grammars are not always easy to come by, and that a user-friendly generator should provide the user with the possibility of exploiting sample XML documents (which are a lot easier to provide) as input to the generator. We consider this to be a fourth property (in addition to the three properties mentioned above), necessary for the usefulness of automatic XML generators.

III. OUR XML DATA GENERATOR

In this paper, we introduce an XML by example data generator, XBeGene that i) fulfills the *dataset usefulness* properties [18] discussed in the previous section (where none of the existing generators [16], [17], [18] complies with all three properties), and ii) allows the user to provide her own sample XML documents as input to the generation process (in comparison with intricate – less user-friendly – grammar-based generators [17], [18], cf. Table I).

Our XML data generation process consists of two phases: i) feature extraction from real XML data, and ii) synthetic XML documents generation.

A. XML Feature Extraction

To generate synthetic XML data that closely resemble a real XML document collection, we start by extracting features that characterize the document collection, such as the XML structure, element/attribute occurrence probabilities and element/attribute value distributions. In the following, we describe the methods for extracting each feature in greater detail.

1) Data structure: Efficient information extraction is difficult if the entire structure of the data cannot be understood [24], [25]. First, the whole structure must be extracted from real data. To obtain a simple and useful description of the data structure of an XML document collection, we use data-guides [25], dynamically generated and maintained structural summaries for semi-structured databases.

The data-guide structure has been originally developed for the OEM data model [25]. Yet, transferring the notion of data-guide to the XML data model is straightforward. A *data-guide DG_C for a collection C* of XML documents is a labeled directed graph structure fulfilling the following conditions:

- Each path in *C* exists in DG_C,
- Each path in DG_C exists in *C*,
- Paths in DG_C are unique (which is not the case in *C*).

In case a common root for the document collection *C* does not exist, a virtual root node is added in constructing DG_C. Every leaf node of the tree is annotated with the IDs of the document leafs it represents (i.e. those reached by the same label path as the index node). Note that a data-guide DG_C does not reflect the node ordering relations in the XML documents in *C*.

While data-guides are less expressive than XML grammars (they do not support cardinality and alternativeness constraints, e.g., ?, *, +, in DTDs, or *MinOccurs* and *MaxOccurs* in XSDs), they can be extracted and handled much more efficiently. Existing approaches to automatically generate an XML grammar from a collection of XML documents are typically NP-Complete [26], which is not practical in many applications, particularly in the context of our study which requires the fast processing of large collections of documents. Instead, we adopt XML data-guide extraction (Figure 1) which is reasonably more efficient, i.e., of polynomial complexity, and thus better suited for the task at hand.

Our algorithm takes as input a collection *C* of parsed XML documents, and generates the corresponding data-guide structure DG_C. For every XML document, the algorithm goes through every element, and verifies whether the corresponding start tag event already exists in the main data-guide HashTable, otherwise it is added. Consequently, it verifies whether each outgoing edge from the current element node (to its children nodes) has already been accounted for in the data-guide, so as to otherwise add it.

Consider for instance the sample XML documents in Figure 2, extracted from the DBLP collection [12] (element values are omitted here for clearness of presentation, since we only address XML structure in this section). The corresponding data-guide graph is shown in Figure 3.

2) Elements and attributes occurrence probabilities: The occurrence rates of XML elements and attributes are typically biased in real-world XML document collections. This is a central piece of information required to better mirror the properties of real-XML data in synthetically generated documents. To reflect the bias in the occurrence count of each element/attribute in the generated data, we must obtain not only the number of distinct element/attribute tag names (such as with existing methods [13], [17]) but also their

```
Algorithm: XML_DG_Extract

Input: XML document collection C
Output: DataGuide graph DGc

Variables:
    HashTable targetHash;
    int currentNodeID;
    stack s;

Begin
1    For each XML document D_i ∈ C
2    {
3        while (parse(D_i).state!=EndDocument){
4            if (parse(D_i).state=StartDocument){
5                currentNodeID=rootID;
6            }
7            else if (parse(D_i).state=StartTag){
8                tagID=targetHash.Lookup(tagName);
9                if (tagID!=null){
10                   if (an edge does not already exist from currentNodeID to tagID) {
11                       graph.addEdge(currentNodeID, tagID);
12                   }
13               }else{
14                   targetHash.insert(tagID, tagName);
15                   graph.addEdge(currentNodeID, tagID);
16                   s.push(currentNodeID);
17                   currentNodeID=tagID;
18               }
19           }
20           else if(parse(S_i).state=EndTag){
21                   currentNodeID=s.pop();
22           }
23       } // End while
24   } // End for each S_i
25   Return DGc;
End
```

Figure 1. XML_DG_Extract Algorithm.

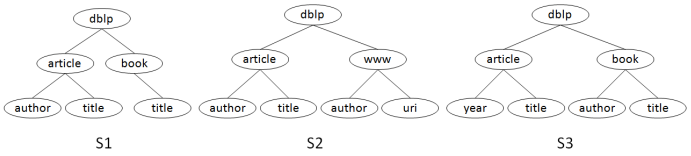


Figure 2. Sample XML documents.

occurrences probabilities, which can only be understood from analyzing the real data collection at hand.

In our approach, we estimate and record the occurrence probabilities of each distinct element and attribute in the real XML data collection *C*, and append probability scores to the corresponding element/attribute representatives in data-guide DG_C in order to provide a unified descriptive structure for the whole dataset. The weighted data-guide is thereafter denoted $\overline{DG_C}$ (Figure 4) . Note that attributes are processed as children of their encompassing nodes, their occurrence probabilities being estimated accordingly. Consider for instance the XML documents in Figure 2. Here, two out of three *article* elements encompass an *author* sub-element. Hence, the occurrence probability of element *author*, as a child of element *article* is equal to 2/3(=66.7%). Figure 5 presents the result of calculating occurrence probabilities for all elements, in all three documents.

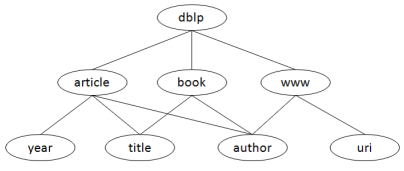


Figure 3. data-guide $\overline{DG_C}$, for $C = \{S_1, S_2, S_3\}$ in Figure 2.

```
Algorithm: Occurrence Probability

Input: XML document collection C
       DataGuide DG_C;
Output: weighted DataGuide graph  DG_C

Variables:
    int currentNodeID;
    int parentNodeID;
    int[] count;
    int[][] check;

Begin
1    For each XML document D_i ∈ C
2    {
3        while (parse(S_i).state!=EndDocument){
4            if (parse(S_i).state=StartTag){
5                currentNodeID = DG_C.get(parse(S_i).getName());
6                if( count[patrentNodeID]!=check[parentNodeID][currentNodeID]){
7                    count[currentNodeID]++;
8                    check[parentNodeID] [currentNodeID] = count[parentNodeID];
9                }
10           }
11           parse(S_i).state =parse(S_i).NextState();
12       } // End while
13
14       for( parentID : rootID .destNodeSet() ){
15           for(childID : parenID.destNodeSet() ){
16               DG_c.probability[parentID][currentNodeID]
17                   =(count[currentNodeID] / count[parentID]);
18           }
19       } // End for
20
21   } // End for
22
23   Return  DG_c;
End
```

Figure 4. Elements/Attributes Occurrence Probabilities Algorithm.

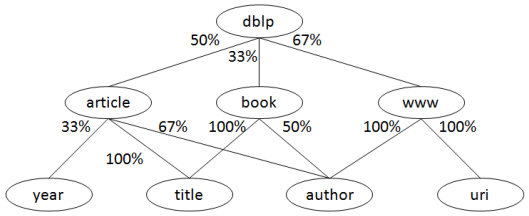
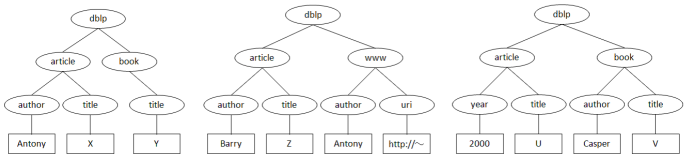


Figure 5. Occurrence Probabilities, i.e., weighted data-guide $\overline{DG_C}$ for the sample document collection $C = \{S_1, S_2, S_3\}$ in Figure 2.

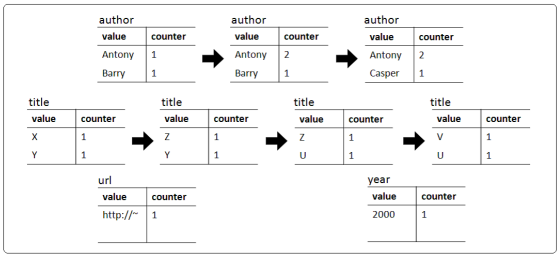
3) Elements and attributes value distributions: In addition to mirroring the structural properties of the original XML documents (using the data-guide structure) and element/attribute occurrence probabilities, we expect our generated XML data to reflect the contents (i.e., element/attribute values) of the real (input) data. In existing XML generators (XMLGen [13], ToXGene [17]), users are required to specify element data-types, which are then exploited by the system to generate the values. Instead, in order to reflect real XML contents, we propose to select data values based on value lists extracted from the input documents, corresponding to each element in the real data collection. In other words,

```
Algorithm: Top-K_SS

Input: XML element E
Output: Value Set V_E fof element E

Begin
1    for each value in E
2    {
3         if(value is monitored){
4              count[i]++;
5         }else if(| V_E |<k){
6              V_E.add(value);
7              count[i]=1;
8         }else{
9              V_E.replace(min, value);
10             count[i]=count[min]+1;
11             V_E.add(value);
12             V_E.delete(min);
13        }
14   }
15   Return V_E;
End
```

Figure 6. Space-saving algorithm, for identifying element/attribute value distributions.



a. XML documents (cf. Figure 2) with sample element values



b. XML element value distributions following algorithm *Top-K_SS*.

Figure 7. Computing value distributions for sample XML documents.

synthetic element values are defined by the vocabularies corresponding to their peers in real documents, such as incorrect values will not appear in the generated data.

We introduce a space-saving algorithm [27] to identify the top-*k* element values in a data stream (Figure 6) . It allows the user to specify the *k* parameter, and consequently identifies the most frequent values within the top-*k* range. A number *k* of pairs of values and counters, i.e.,*(value, count)*, are stored, and initialized by the first *k* distinct values and their exact counts. Value counts are consequently incremented following their occurrences in the considered element/attribute value sequences.

Consider the sample XML dataset in Figure 7.a. Frequent top-*k* element values for corresponding XML elements are shown in the rightmost tables of Figure 7.b.

B. Synthetic XML Document Generation

We use the information obtained through analysis of real XML data (feature extraction phase) for synthetic XML documents generation. Having evaluated the various structural and content properties of the input XML data (feature extraction phase), we consequently exploit these information for generating synthetic XML documents. Given a real data collection, the input to the data generation process consist of two main XML-based files: one to describe the structure and occurrence probabilities of XML elements/attributes, i.e., the weighted data-guide $\overline{DG_C}$ corresponding to document collection *C*, the other to describe element/attribute value distributions. Furthermore, the user can choose between two methods for specifying data size: the total number of XML elements/attributes, or the number of data bytes in the resulting file. Our generator outputs synthetic XML documents that mirror the various structural and content features of real input documents.

Our algorithm for XML data generation, entitled XBeGene (XML By example Generator), is shown in Figure 8. XMLGene starts by selecting elements such as their occurrence probabilities are most similar to those of real data (lines 7-8). It compares the occurrence probability of each element in the generated XML dataset to that of its peer in the real data collection. Elements with the highest probability discrepancies (between the generated data and real data), are gradually selected and added to the generated dataset (lines 9-14). Note that the number of added elements is adjusted to maintain, not only the occurrence probability the added element itself, but also the probability of occurrence of its corresponding parent element (lines 15-23). This process is repeated until the generated dataset reaches the specified data size.

A sample synthetic XML document generated using algorithm *XBeGene* is shown in Figure 9.a. It is based on the XML document set used in our running example (cf. Figure 8), which structure and element occurrence probabilities (i.e., weighted data-guide) are reported in Figure. 9.b, and value distributions are reported in Figure. 9.c. For example, element *book* appears twice in the synthetic document (Fig. 9.a.), which amounts to an occurrence probability of 33% (considering its siblings). This corresponds exactly to the same occurrence probability in real data (Fig. 9.b). In addition, only one of the two *book* elements contains a sub-element *author*, since our generator reflects the occurrence probability of the *book* element taking into account the occurrence frequencies of its sub-elements, e.g., element *author* in real data (Fig. 9.c).

IV. Experimental Evaluation

This section presents our XBeGene prototype system for XML feature extraction and synthetic documents generation, as well as the battery of tests conducted to evaluate and validate our approach. The XBeGene system is implemented in *Java 6* language as a desktop application with robust GUI, built on the top of *NetBeans 6.8* to allow cross platform usage. It consists of two main components, for i) XML

```
Algorithm: XBeGene

Input:  DG_i : Weighted data-guide to describe XML structure and element/attribute occurrence probabilities
        VD : Element/Attribute value distributions file
        DocSize : Synthetic Document Size
        NbDocs : Number of Synthetic Documents
Output: generated XML data OutputFileCollection

Begin
 1    For  i=1 to NbDocs
 2    {
 3        Create SynDoc_i
 4        SynDoc_i.add(Root of DG_i)
 5        int max, dif = 0;
 6        Element El = ϕ;
 7        while(Doc_i.size < DocSize)
 8        {
 9            for each element e in children of SynDoc_i.Root
10            {
11                dif = |OccProb(e)_RealData − OccProb(e)_SynData|
                  // difference between occurrence probability of e in real data
12                  and occurrence probability of e  in synthetic data
13                if(max < dif){
14                    max = dif;
15                    El = e;
16                }
17            }
18            output(SynDoc_i, El);
19            for each f included in children of El
20            {
21                int mul = NbOcc(El) * OccProb(El/ f∈ El);
                  // multiply number of occurrence of El in synthetic data
                     by occurrence probabilities of El containing f  in real data
22                if ( mul − NbOcc(El/ f∈ El)_SynData >= 0.5)
23                {
24                    output(SynDoc_i, f );
25                    ValueSet = VD.getValueSet(f);
26                    value = ValueSet.getValue();
                      // value is chosen from ValueSet following corresponding element value distribution
27                    output(SynDoc_i, value);
28                }
29            }
30        }   // End while
31    }  // End for
32    Return OutputFileCollection;
End
```

Figure 8. Algorithm XBeGene for synthetic XML document generation based on real data.

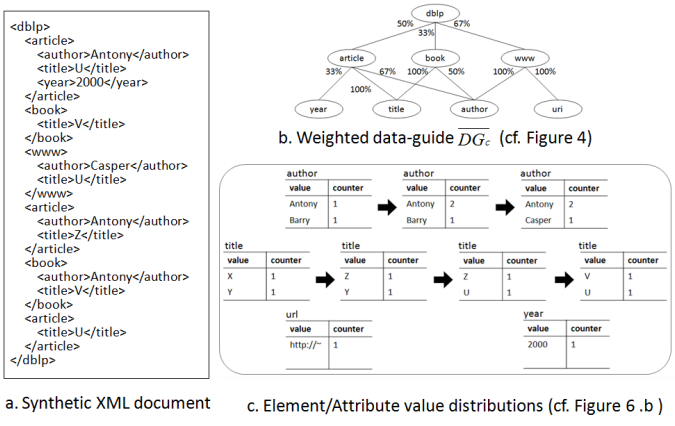


Figure 9. Example of Output XML data.

feature extraction and ii) XML data generation. The feature extraction component accepts as input a (set of) real XML document(s), and outputs corresponding structural properties (data-guide), element/attribute occurrence probabilities (data-guide node weights), and element/attribute value distributions. The feature descriptions are consequently exploited as input to the XML data generator component, along with the number of synthetic documents and the maximum document size specified by the user. A simplified activity diagram describing the XBeGene prototype is depicted in Figure 10.

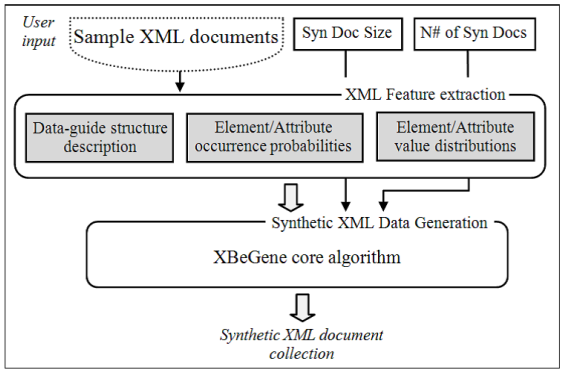


Figure 10. Simplified activity diagram describing *XBeGene*.

We conducted various experiments to test and validate our approach, in comparison with one of its most sophisticated alternatives: ToXGene [17]. We first verify the system's accuracy in generating synthetic documents which closely mirror the characteristics of real data. To do so, we estimate the similarity between real and synthetic XML documents using XML document clustering. Second, we evaluate the correlation in element occurrence probabilities and value distributions between the generated and real data, in order to verify the similarity (in number of occurrences) between each pair of matching elements. All tests were executed on a standard PC with Intel Core 2 Duo CPU (3.06GHz), 8GB RAM, and Windows 7 OS . Our tests are based on the bibliography of SIGMOD Record [11], DBLP [12](532 MB) and XMLGen [16]. Experimental results confirm our algorithm's effectiveness and efficiency in generating synthetic XML documents of arbitrary sizes, in comparison with ToXGene [17], while preserving the structural and content features of user data.

A. *Accuracy*

Our first experiment measures the similarity between real XML data and the generated synthetic data, in order to evaluate XBeGene's efficiency in mirroring the characteristics of user input XML documents. We provide as input a number of heterogeneous documents, generate corresponding output documents, and consequently compare both input and output document sets. We exploit document clustering to assess the similarity between (real) input and (synthetic) output document sets. The main idea is to verify whether the clustering result of input documents correlates with the clustering result of output documents. In our experiment, we utilize an agglomerative hierarchical clustering algorithm, built upon an XML path similarity model to evaluate the similarity between XML documents [28]. The approach developed in [28] provides an improved path similarity approach taking into account the main properties of XML documents in the comparison process, while preserving the same complexity levels as existing path-based methods. Similarity between real and synthetic XML data is evaluated based on clustering

accuracy:

$$accuracy = \frac{number\ of\ documents\ correctly\ clustered}{total\ number\ of\ documents}$$

In other words, the closer input and output clusters are, the higher the similarity between real and generated XML documents, and thus the higher the accuracy of the XML data generation process in preserving the characteristics of the user input XML data. We utilize as input data real XML documents from the online SIGMOD Record collection [11], and compare our system's results to those generated by ToXGene [17]. First, we perform clustering on the real document collection and identify the input clusters $C_1; C_2; :::; C_n$. Second, we perform clustering on the synthetic documents generated by XBeGene (based on the SIGMOD collection) and by ToXGene, and identify the output clusters $S_1; S_2; :::; S_n$. The size of the real document collection, i.e., sum of the sizes of all input documents, is denoted N (the value of N differs depending on the input document collection at hand), such as the size of the generated data varies w.r.t. N; $N_{Syn}1 = 1 \times N, N_{Syn}2 = 2 \times N, N_{Syn}3 = 3 \times N, N_{Syn}5 = 5 \times N$). Consequently, we verify how closely clusters C_i correspond to clusters S_j, correlating clustering results by evaluating accuracy. Figure 11 and Figure 12 depict the accuracy of XBeGene w.r.t. the number of clusters and document collection size respectively. Figure 11 depicts clustering accuracy (y-axis) w.r.t. the number of real clusters (x-axis). For instance, when the number of real clusters is equal to 3, synthetic documents are distinguished by 3 different XML grammars corresponding to 3 real XML data sets. Experimental results show that accuracy levels remain almost steady (varying from 0.7 to 1.0) when increasing the number of real clusters. Similarly, accuracy remains steady when varying the size of the synthetic document collection (cf. Figure 12). Therefore, results show that XBeGene generates synthetic XML documents of various sizes which are highly similar to the input XML documents. In addition, Figure 13 depits the average accuracy levels attained using our method and ToXGene [17]. Figure 14 depicts clustering average accuracy (y-axis) of our method and ToXGene. Our approach steadily outperforms its predecessor, regardless of the number of clusters considered and document collection size. Recall that ToXGene underlines limited capabilities in controlling the structure of the generated XML data, which explains the poor accuracy results.

B. Occurrence Probabilities and Value Distributions

In addition, we compare element occurrence probabilities and value distributions between synthetic and real XML documents in order to verify the similarity between each pair of matching elements. We generate synthetic documents of different sizes (60MB and 270MB) based on the XMark document (115MB) and calculate the respective element

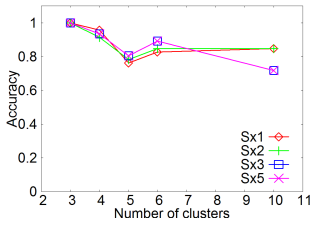


Figure 11.
Accuracy of clustering (x-axis=number of clusters).

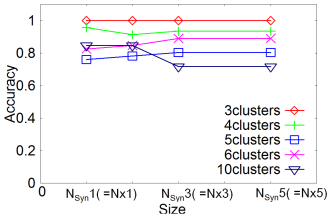


Figure 12.
Accuracy of clustering (x-axis=size).

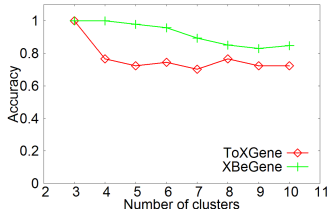


Figure 13.
Comparing average accuracy between XBeGene and ToXGene (x-axis=number of clusters).

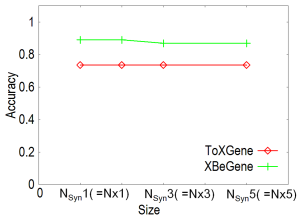


Figure 14.
Comparing average accuracy between XBeGene and ToXGene (x-axis=size).

occurrence probabilities for synthetic documents. In same way, we generate a synthetic document using ToXGene and calculate the each element occurrence probability. Then, we compare the element occurrence probabilities of XMark, ToXGene and XBeGene. Table II contrasts the average scores in the XMark document, and the synthetic documents collection generated based on XMark by using ToXGene and XBeGene. Results in Table II show high correlation levels for all elements in XMark and the documents generated using XBeGene, regardless of the synthetic document collection size. While ToXGene provides advanced settings to capture element contents, however, it provides limited control over the XML data structure, which explains the lower correlation levels in Table II. In other words, results concur and confirm the high accuracy levels obtained in our clustering experiments described in the previous paragraph.

Note that we have also assessed the overall throughput of XBeGene, considering different sizes of generated documents, in order to evaluate execution time. Results show that XBeGene's time grows almost linearly w.r.t. synthetic document size (and is lower, on average, than ToXGene's

Table II
AVERAGE OCCURRENCE PROBABILITIES FOR EACH ELEMENT IN THE DATA COLLECTION[%]

element	XMark 115MB	XBeGene 60MB	XBeGene 270MB	ToXGene 60MB
/africa/item/mailbox/mail	62.545	62.545	61.818	45.455
/asia/item/mailbox/mail	60.250	60.250	60.500	45.000
/australia/item/mailbox /mail	60.455	60.455	58.778	49.545
/watches/watch	88.350	88.350	88.350	78.500
/open_auction/bidder	90.300	90.300	100.000	82.670
/categories/description/text/bold	5.400	6.000	5.9000	36.240
/close_auction/description /emph	5.700	5.700	5.700	30.000

execution time). Detailed complexity analysis and timing graphs have been omitted due to lack of space.

V. Conclusions

In this paper, we introduced an XML-by-example generator (XBeGene) for producing synthetic XML documents which closely satisfy the structural characteristics and content distributions of user-defined input data. Our method uses weighted data-guides as an efficient means to describe the structural features of the input document collection, and a dedicated space-saving algorithm for assessing element/attribute value distributions. Our theoretical study as well as our experimental evaluation showed that our approach is efficient and scalable in generating XML documents of arbitrary sizes which mirror the structural and content features of user provided real document collections.

As continuing work, we are currently extending of our approach to enable the generation of element/attribute values and/or tag names according to different controlled vocabularies, taxonomies and/or thesauri. We believe the hierarchical relations in taxonomies/thesauri can be extremely useful in highlighting the structural characteristics of XML data. In addition, we plan to consider simplified XML grammars, relaxing certain DTD/XSD constraints, in order to obtain an XML structural representative which is i) more descriptive than data-guides, and ii) which could be handled in average polynomial time. In the near future, we plan to extend our generator for domain-specific applications such as CML to generate synthetic collections of protein sequences (useful for knowledge management in bioinformatics [22]), and SVG to generate synthetic collections of Web images.

Acknowledgment

The XML clustering method is provided by Cheikh Bedy Mohamed (Shizuoka University, Japan).

This research was partially supported by Scientific Research(B)(19300026), and *Japan Society for the Promotion of Science (JSPS)* 2010 research fellowship n.PE10006.

References

[1] J. Tekli, R. Chbeir, K. Yetongnon, ”A Hybrid Approach for XML Similarity,” SOFSEM (1), 783-795, 2007.

[2] J. Tekli, R. Chbeir, K. Yetongnon, ”Extensible User-Based XML Grammar Matching,” ER 2009, 294-314.

[3] S. Helmer, ”Measuring the Structural Similarity of Semistructured Documents Using Entropy,” *in* Proceedings of the International Conference on Very Large Databases (VLDB), 2007, pp. 1022-1032.

[4] G. Cobena, S. Abiteboul and A. Marian, ”Xydiff, tools for detecting changes in XML documents,” 2001. [Online]. Available: http://www.inria.fr/?cobena/XyDiffWeb/

[5] G. Cobena, S. Abiteboul, A. Marian, ”Detecting Changes in XML Documents,” *in* Proceedings of the IEEE International Conference on Data Engineering (ICDE), 2002, pp. 41-52.

[6] E. Bertino, G. Guerrini, M. Mesiti, ”A Matching Algorithm for Measuring the Structural Similarity between an XML Documents and a DTD and its Applications,” *in* Elsevier Computer Science, 2004, (29):23-46.

[7] L. Candillier, I. Tellier and F. Torre, ”Transforming XML Trees for Efficient Classification and Clustering,” *in* Proceedings of the Workshop of the Initiative for the Evaluation of XML Retrieval (INEX), 2005, 469-480.

[8] T. Dalamagas, T. Cheng, K. Winkel and T. Sellis, ”A Methodology for Clustering XML Documents by Structure,” *in* Information Systems, 2006, 31(3):187-228.

[9] A. Nierman and H. V. Jagadish, ”Evaluating structural similarity in XML documents,” *in* Proceedings of the ACM SIGMOD International Workshop on the Web and Databases (WebDB), 2002, pp. 61-66.

[10] J. Bosak, The Plays of Shakespeare in XML, 1999. [Online]. http://xml.coverpages.org/bosakShakespeare200.html

[11] SIGMOD Record Document Collection. [Online]. Available: http://www.sigmod.org/record/xml/

[12] The DBLP Computer Science Bibliography. [Online]. Available: http://www.informatik.uni-trier.de/ ley/db/

[13] A. Schmidt, F. Waas, M. Kersten, M. Carey, I. Manolescu and R. Busse, ”XMark: a Benchmark for XML Data Management,” Proceedings of the International Conference on Very Large Data Bases (VLDB), 2002, pp. 974-985.

[14] B. Yao, M. Ozsu and N. Khandelwal, ”XBench: Benchmark and Performance Testing of XML DBMSs,” Proceedings of the International Conference on Data Engineering (ICDE), 2004, pp. 621-633.

[15] K. Runapongsa , J. Patel, H. Jagadish, Y. Chen, S. Al-Khalifa, ”The Michigan Benchmark: towards XML Query Performance Diagnostics,” Information Systems, 2006, 31(2): pp. 73-97.

[16] A. Aboulnaga, J. Naughton and C. Zhang, ”Generating Synthetic Complex-Structured XML Data,” *in* Proceedings of the ACM SIGMOD International Workshop on the Web and Databases (WebDB), 2001, pp. 79-84.

[17] D. Barbosa, A. Mendelzon, J. Keenleyside, K. Lyons, ”ToXgene: an Extensible Template-based Data Generator for XML,” Proceedings of the ACM SIGMOD International Workshop on the Web and Databases (WebDB), 2002, pp. 49-54.

[18] S. Cohen, ”Generating XML Structure Using Examples and Constraints,” Proceedings of the Very Large Data Bases Endowment (PVLBD), 2008, 1(1):490-501.

[19] T. J. Moran, ”Impact Of XML On Data Interchange: An XML/EDI Model,” Business Information Systems, 2007, 11(3): 35-42.

[20] S. Amer-Yahia, J. Shanmugasundaram, ”XML Full-Text Search: Challenges and Opportunities,” Proceedings of the International Conference on Very Large Data Bases, 2005, Tutorial Slides, http://www.vldb2005.org/program/slides/fri/s1368-amer-yahia.ppt.

[21] F. Pereira, ”Technologies for Digital Multimedia Communications: An Evolution Analysis of MPEG Standards,” China Communications Journal, 2006.

[22] S. Asak, V. S. Batra, D. N. Bhardwaj, P. V. Kamesam, P. Kankar, M. P. Kurhekar, B. Srivastrava, ”A System for Knowledge Management in Bioinformatics,” Proceedings of the International Conference on Information and Knowledge Management (CIKM), 2002, pp. 638-641.

[23] M. Nicola, I. Kogan, B. Schiefer, ”An XML Transaction Processing Benchmark,” Proceedings of the ACM International Conference on Management of Data (SIGMOD), 2007, pp. 937-948.

[24] R. Goldman, J. McHugh, J. Widom, ”From Semistructured Data to XML: Migrating the Lore Data Model and Query Language,” Proceedings of the ACM SIGMOD International Workshop on the Web and Databases (WebDB), 1999, pp. 25-30.

[25] R. Goldman, J. Widom, ”data-guides: Query Formulation and Optimization in Semistructured Databases,” Proceedings of the International Conference on Very Large Databases (VLDB), 1997, pp. 436-445.

[26] M. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, K. Shim, ”Xtract: A system for extracting document type descriptors from XML documents,” Proceedings of the ACM International Conference on Management of Data (SIGMOD), 2000, pp. 165-176. Dallas, Texas, USA.

[27] A. Metwally, D. Agrawal, A. E. Abbadi, ”Efficient computation of frequent and top-k elements in data streams,” International Conference on Database Theory (ICDT), 2005, pp. 398-412.

[28] C. B. Mohamed, S. Yokoyama, N. Fukuta, H. Ishikawa, R. Chbeir, ”New Approach for Computing Structural Similarity between XML Documents,” Master’s thesis, Shizuoka University, Japan, 2010.