ToXgene: An extensible template-based data generator for XML

Denilson Barbosa[1] Alberto Mendelzon[1] John Keenleyside[2] Kelly Lyons[2]

[1] Department of Computer Science
University of Toronto
{dmb,mendel}@db.toronto.edu

[2] IBM Toronto Lab
{keenley,klyons}@ca.ibm.com

Abstract

Synthetic collections of XML documents are useful in many applications, such as benchmarking (e.g., XMach-1, Xmark), and algorithm testing and evaluation. We present ToXgene a template-based generator for large, consistent collections of synthetic XML documents. Templates are annotated XML Schema specifications describing both the structure and the content of the data to be generated. Our tool was designed to be declarative, and general enough to generate complex XML content and to capture most common requirements, such as those embodied in current benchmarks. The paper gives an overview of the ToXgene template specification language and the extensibility of our tool; and reports preliminary experiments with ToXgene carried out at the IBM Toronto Lab, which show that our tool can closely reproduce the data sets for the Xmark and the TPC-H benchmarks.

1 Introduction

Synthetic collections of XML documents have many applications in benchmarking, testing and evaluating various algorithms, tools and systems. Moreover, different applications require different documents, with different complexities, sizes, etc. For instance, a benchmark for data-intensive applications might require a large and relatively homogeneous document, with many references among elements, while an adequate test suite for a parser might be an heterogeneous collection with thousand of documents with varying sizes. Given the complexity of writing and/or customizing hard-coded synthetic data generators for specific scenarios, we believe a declarative tool for generating synthetic XML documents will prove useful.

ToXgene is a template-based tool for generating large, consistent synthetic collections of complex XML documents. The ToXgene template specification language is a subset of the XML Schema [5] notation augmented with annotations for specifying certain properties of the intended data, such as probability distributions, the vocabulary used for generating `CDATA` content, etc.

We give an overview of our template specification language, and the kinds of XML content our tool can generate; we discuss how our tool can be extended to generate data for specific domains; and we report on experiments carried out at the IBM Toronto Lab, showing that ToXgene can closely reproduce the characteristics of synthetic data used in existing benchmarks. This work is part of the ToX project, recently started at the University of Toronto; more on ToX can be found in [2].

1.1 Related work

A general purpose generator for synthetic XML documents is presented in [1]. Our work differs from that in the following ways. First, the data generation process in centered on a conceptual description of the intended data (a template). Thus, ToXgene gives the user total control over the data to be generated, unlike the method in [1], where both the structure and the content of the documents are randomly generated. We note that our tool allows some controlled randomness on input templates, thus it can generate documents with fairly irregular structure. Second, our tool generates more complex XML content, including elements with mixed content; attributes; non-gibberish text; and different numerical and date values. Also, our tool supports different probability distributions.

The IBM XML Generator [7] is also a declarative tool, and is similar to ToXgene in the sense it also requires a conceptual description of the data to be generated: a annotated Document Type Definition (DTD) specifying the structure and the characteristics of the data. There are many limitations to that tool, however. For instance, it allows one to *limit* the maximum depth of the document tree, or the number of `ID` and `IDREF` attributes in the documents, but it does not allow one to *specify* actual values for these properties. Moreover, unlike ToXgene, that tool does not allow the use of different probabilities of occurrence on a *per element* basis, nor the generation of `CDATA` content of different datatypes (e.g., strings, dates, etc.). On the other hand, the IBM generator deals with certain aspects of the XML standard that are not addressed in ToXgene, such as `ENTITY` declarations and processing instructions [4].

Furthermore, our tool differs from both approaches above in the following ways. First, ToXgene allows element sharing, i.e., the values of some elements (or attributes) can be shared among different elements (or attributes) in the same (or in different) XML documents. This allows the generation of collections of correlated documents (i.e., documents that can be *joined* by value). Second, ToXgene can produce data conforming to user-specified integrity constraints. Most common constraints are supported; for example, our tool allows the generation of consistent references within and across documents. Third, ToXgene allows the use of existing data when generating documents; thus, one can grow an existing collection of synthetic documents while maintaining its consistency, or mix real and synthetic data in the generation process (e.g., use real country names as in XMark). Finally, our tool can be easily extended to allow the generation of domain-specific content, whenever the built-in tools are insufficient.

Annotated XML Schema specifications have also been used for defining relational storage methods for XML documents [3].

The remainder of the paper is organized as follows. Next section presents an overview of the template specification language and illustrates the kinds of documents ToXgene can produce. The extensibility mechanisms of ToXgene are discussed in Section 3 and a summary of experimental results with ToXgene are presented in Section 4; a more detailed analysis of the experiments is given in the appendix. We conclude in Section 5.

2 ToXgene template specification language

Among the various languages for specifying XML content, we chose XML Schema as the basis for our template language for two reasons: it is a W3C standard, thus it is expected to become familiar to XML practitioners; and it allows a more detailed description of XML content than DTDs. In particular, it allows the specification of types, for describing literals (i.e., `CDATA` content), elements and attributes. However, we note that having an XML Schema specification alone is not enough for generating useful synthetic data; at the very least, we need to annotate the schema for specifying which type should be used as the type for the root element of the documents to be generated.

We also define annotations for other purposes, such as specifying probability distributions of occurrences of elements and attributes; defining element sharing; defining integrity constraints over XML elements; and providing some controlled randomness in the structure of the data to be generated. More details on the ToXgene template language can be found in [8].

Types and genes. A *type* is a specification of some valid XML content, and can be either a `simpleType` or a `complexType` [5]; *instances* of a `simpleType` are `CDATA` literals, while instances of a `complexType` are either XML elements, `CDATA` literals, or a mix of both.

A *gene* is a specification of either an element (an *element* gene) or an attribute (an *attribute* gene), and contains a name and a type; an *instance* of an element gene with name n and type t is an element whose opening and closing tags are labeled with n and whose content is an instance of t; similarly, an *instance* of an attribute gene with name n and type t is an is an attribute whose name is n and whose content is an instance of t. An attribute gene can only be declared within a `complexType`, and its type must be a `simpleType`. As we will see in the examples, the genes are the building blocks of templates.

2.1 Specifying probability distributions, types and *genes*

Figure 1 contains the examples we use to illustrate the discussion in this section. All ToXgene-specific annotations are prefixed by the keyword `tox` and presented in a different font.

```
<tox-distribution name="c1"
  type="exponential" minInclusive="5"
  maxInclusive="100" mean="35">
```

(a) Declaration of an exponential probability distribution.

```
<simpleType name="isbn_type">
  <restriction base="string">
    <pattern value="[0-9]{10}"/>
  </restriction>
</simpleType>

<simpleType name="my_float">
  <restriction base="float">
    <tox-number tox-distribution="c1"/>
  </restriction>
</simpleType>
```

(b) Specification of a `simpleType`.

```
<complexType name="my_book">
  <attribute name="isbn" type="isbn_type"/>
  <element name="title" type="string">
    <tox-string type="text" maxLength="30"/>
  </element>
  <element name="author" type="my_author"
    minOccurs="1" maxOccurs="5"/>
  <element name="price">
    <complexType mixed="true">
      <attribute name="currency">
        <simpleType>
          <restriction base="string">
            <tox-value>CDN</tox-value>
          </restriction>
        </simpleType>
      </attribute>
      <tox-number minInclusive="5"
      maxInclusive="100"/>
    </complexType>
  </element>
</complexType>
```

(c) A `complexType` declaration.

Figure 1: Notation for specifying types, genes and probability distributions in ToXgene.

Specifying probability distributions. A probability distribution is declared using the **tox-distribution** annotation, as shown in Figure 1(a), and referenced via its name, as shown in Figure 1(b). A single template might specify multiple probability distributions, which can be used to determine, for example, the number of occurrence of elements and attributes, the length of string literals, and instances of numerical types, as in Figure 1(b). ToXgene currently supports the uniform, normal, exponential and log-normal distributions, as well as arbitrary discrete distributions, where the user provides all possible outcomes together with their respective probabilities of occurrence. In order to allow the static checking of the consistency of the templates, we require each probability distribution to have a maximum and a minimum value; all values outside this interval have null probability of occurrence.

Specifying `simpleType`s. A `simpleType` is a specialization of a *base* type, e.g., string, integer, etc.; the *domain* of a `simpleType` is a subset of the domain of the base type it is built upon; and an instance of a `simpleType` is an element chosen from its domain. For example, the `isbn_type`, specified in Figure 1(b), is a specialization of the string base type; its domain is the set of strings that conform to the given pattern; and `1234567890` is one of its instances. `simpleType`s are the basic content specification tools for defining genes.

The **tox-number**, **tox-string** and **tox-date** annotations are used to refine a type definition (e.g., to specify that instances of `my_float` in Figure 1(b) obey the probability distribution **`c1`**, and that the content of instances of the `title` element in Figure 1(c) are non-gibberish strings[1], no more than 30 characters long), or to specify the generation of elements with mixed content (e.g., the `price` element in Figure 1(c)). We also provide the **tox-value** annotation, for specifying constants.

Specifying `complexType`s and genes. A `complexType` specification (see Figure 1(c)) contains definitions of element genes, attribute genes, or annotations defining `CDATA` literals, required for defining elements of mixed content. Both named and anonymous types are allowed in ToXgene, as shown in the figure. Our tool supports all element content models defined in [4]: character data, as in the `isbn` element; elements, as in the `book` element; and mixed, as in the `price` element.

[1]Our tool supports two built-in string types: **gibberish**, and **text**, as defined in [9]. However, ToXgene allows can produce strings using a different vocabulary, as discussed in the appendix.

```
<tox-list name="book_list" unique="[book/isbn]">
  <element name="b_rec" minOccurs="200">
    <element name="isbn" type="isbn_type"/>
    <element name="author_id" type="integer"
      maxOccurs="5">
      <tox-sample path="[author_list/author]"
        duplicates="no">
        <tox-expr value="[id/!]"/>
      </tox-sample>
    </element>
  </element>
</tox-list>
```

(a) Declaration of a list with 200 book records. Each record has a unique ISBN value and up to 5 `author_id` values, sampled (without repetition) from a list of authors, declared elsewhere.

```
<element name="book" minOccurs="200">
  <complexType>
    <tox-scan path="[book_list/b_rec]" name="a">
      <attribute name="isbn" type="isbn_type">
        <tox-expr value="[isbn/!]"/>
      </attribute>
      <element name="title" type="string"/>
      <attribute name="authors" type="IDREFS"
        tox-maxOccurs="unbounded">
        <tox-scan path="[$a/author_id]">
          <tox-expr value="'author'#[!]"/>
        </tox-scan>
      </attribute>
    </tox-scan>
  </complexType>
</element>
```

(b) Specifying queries over lists; the symbol # in the expression corresponds to the string concatenation operation.

Figure 2: Defining and querying lists.

2.2 Specifying element sharing and integrity constraints in ToXgene

ToXgene allows *element sharing* both within and across documents; i.e., different elements (or attributes), in the same or in different documents, can have the same `CDATA` content. This allows the generation of collections of *correlated* documents (i.e., documents that can be *joined* by value). In our tool, element sharing is achieved by generating all shared content prior to generating any documents. The shared content is kept in what we call **tox-list**s; such lists are queried at document generation time. Integrity constraints over the contents of a list, such as uniqueness of certain values, can be specified, thus ensuring the consistency of the collections.

Specifying lists. Lists are declared by **tox-list** annotations. Each list has a unique name, an element gene that defines its contents, and, optionally, some integrity constraints. Figure 2(a) shows the specification of a list of books, each containing an ISBN and up to 5 references to author elements, stored in another list. The number of elements in a list is determined by the gene defining it (exactly 200 in the example in the figure). The **unique** constraint in the list in Figure 2(a) ensures that no duplicate ISBN values are stored in the list; more complex integrity constraints can be specified using *where* clauses (see [8] for details).

Querying lists. ToXgene defines a language for specifying expressions that are evaluated at content generation time. This language allows arithmetic and string operations; operands can be the results of queries, instances of `simpleType`s generated "on-the-fly", or constants. Moreover, all expressions are typed, and type checking and casting mechanisms are implemented. Expressions are useful for defining conditional instantiation of genes (see Section 2.4), for specifying where clauses in list definitions and selection conditions on cursors (see below). Due to space limitations, we focus on expressions defining queries only; for details, see [8].

The contents of a list are retrieved using *cursors* and *queries*, both of which are specified using *path expressions*. A path expression is a sequence of *names* separated by `'/'`; a name can be one of: a list name, an element name, a cursor name, or `!`, which is shorthand for `CDATA`. All path expressions are enclosed by squared brackets. The elements over which a cursor iterates are specified by a path expression rooted at a list or at another cursor, and, optionally, a selection condition. Different cursors might iterate over the same content, using different access patterns. ToXgene provides cursors for sequential scans (using the **tox-scan** annotation), and sampling with or without repetition (using the **tox-sample** annotation).

Queries are expressions (declared using **tox-expr** annotations) that extract the actual `CDATA` content of the elements in a cursor. Moreover, queries are always evaluated against the current element of the cursor they reference; by default, queries refer to their closest ancestor cursor declaration. Explicit references to a cursor can be made by

```
<tox-document name="books" copies="1"
  <complexType>
    <element name="books">
      <complexType>
        <element name="book" type="book"
          minOccurs="200" maxOccurs="200">
          <tox-scan path="[book_list/b_rec]">
            ...
      </complexType>
    </element>
  <complexType>
</tox-document>
```

(a) Single file, multiple `book` elements.

```
<tox-document name="book" copies="200"
  <complexType>
    <element name="book" type="book">
      <tox-scan path="[book_list/b_rec]">
      ...
    </element>
  <complexType>
</tox-document>
```

(b) Collection of files, each having a single `book` element.

Figure 3: Declaring single documents and collections of documents in ToXgene.

rooting a path expression at a cursor name (see Figure 2(b)).

Cursors are always declared within genes, and are updated whenever these genes are instantiated. Thus, the queries defining the contents of each of the 200 instances of the `book` element gene in Figure 2(b) will be evaluated against different `b_rec` elements. Therefore, the output of the template in Figure 2(b) will be 200 `book` elements, each with a distinct value for its `isbn` attribute[2]. Cursors can be nested (see Figure 2(b)); the name of the parent cursor (`$a` in the example) is used as the starting node of the path expression defining the nested cursor.

Consider the `authors` attribute gene specified in Figure 2(b); its content is obtained by querying the `author_id` values of the current `b_rec` element in the outer cursor. Recall that up to 5 author id's are generated per book. The **tox-maxOccurs="unbounded"** annotation in the gene specification determines that ToXgene should generate as many instances of the gene as there are elements in the cursor defining its content. Therefore, the resulting attribute will be a comma separated list of author id's.

The ToXgene query language also defines useful aggregate and string manipulation functions.

2.3 Specifying XML documents

Documents are specified by **tox-document** annotations, as shown in Figure 3(a). Similarly to a list, a document specification contains a name and a gene, whose instance will be the *root* element of the document. Collections of documents are specified by declaring the **copies** attribute in a document declaration (see Figure 3(b)).

ToXgene's output for the specification in Figure 3(a) is a single XML document, called `books.xml`, and whose root is an element called `books`, whose children are 200 `book` elements copied from a list. ToXgene's output for the specification in Figure 3(b) is a collection with 200 XML documents, called `book0.xml`, `book1.xml`, etc.; each document has an element called `book` as root, and its contents are copied from the same list. We note that the ith book in `books.xml` corresponds to the `book`i`.xml` document in the collection.

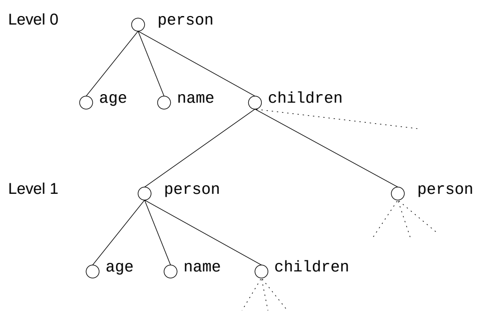
2.4 Specifying irregular structures and recursive elements

ToXgene templates can also have control statements that impose conditions on the instantiation of genes. Each control statement contains one or more *blocks* of genes; at evaluation time, exactly one such block is chosen, depending on conditions specified in the control statement. Only genes in the chosen block are instantiated. We provide IF-THEN-ELSE statements, which can be nested arbitrarily, and also a "lottery" control statement, in which the chosen block is randomly selected, according to a probability distribution. Since different blocks might define completely different genes, the control statements provide some controlled randomness in the structure of the XML documents produced by our tool.

[2]Note that ISBN values are unique in `book_list`.

```
<element name="person" maxOccurs="500"
  tox-recDistribution="r1">
  <complexType>
    <element name="name" type="name_type"/>
    <element name="age" type="age_type"/>
    ...
    <element name="children">
      <complexType>
        <element name="person" maxOccurs="5"/>
      </complexType>
    ...
  </complexType>
</element>
```

(a) Specification of a recursive gene.



(b) Structure of resulting XML elements.

Figure 4: Specification and instantiation of recursive element genes.

Recursion in element genes. ToXgene also allows recursive element generation, as illustrated by Figure 4. The example defines a gene for up to 500 `person` elements, each of which might have other `person` elements as their descendants, under the `children` element. However, each `person` can have no more than 5 `person` elements as their descendants. The depth of the XML tree generated by that gene is determined by distribution `r1`.

3 Extending ToXgene

ToXgene is a general purpose tool, and as such provides built-in tools for generating XML content conforming to the most common datatypes. Although preliminary experimentation with our tool shows it can easily reproduce the synthetic data used in complex benchmarks, using a combination of queries and string operations, ToXgene was designed to allow the use of user-defined `CDATA` generators. For example, one might want to use pseudo-random DNA sequences, generated by a given algorithm. In fact, one is not restricted to using random `CDATA` content; one might want to use readings from a sensor, or the results of queries over the Web, in synthetic XML documents.

ToXgene was implemented in Java 2, and its architecture was designed in a way that Java classes implementing generators of instances of new `simpleType`s could be added with little effort: all that is required is registering the new code in the Gene Factory Module; of course, the added class has to implement the interface defined in our code, which consists of a single method. The Java code that produces instances of `simpleType`s is not intended for producing elements (i.e., strings containing element tags). However, there is nothing in our code that enforces this behavior. Thus, if one needed XML elements with random tag names, it would suffice to encapsulate the data generator in [1] as a new `simpleType` in ToXgene. Our tool can also be coupled with external tools, since it has the ability of storing and reading lists from files.

4 Experimental results

Our goal with ToXgene was the generation of consistent collections of complex XML documents. To test our tool, we reproduced the synthetic data produced by the data generators of the TPC-H (`dbgen`) and the XMark benchmarks (`xmlgen`). For the TPC-H experiment, we generated several XML documents, each corresponding to a relation in the TPC-H database, and loaded the data in these documents into a relational database on DB2 using IBM's XML Extender [6]. The data produced by our tool did not violate any integrity constraints, and the results of the queries in the TPC-H workload using our ToXgene data and `dbgen` data were comparable. For the XMark experiment, we generated a single XML document containing thousands of internal references among elements; our document parsed without problems. We executed some of the queries in the Xmark workload using our data and `xmlgen` data, and obtained comparable results. More details on these experiments can be found in Appendix A.

5 Conclusions and future work

In this paper we introduced ToXgene: an extensible template-based generator for consistent collections of correlated synthetic XML documents. We presented an overview of our template specification language, which is based on XML Schema, and used for describing both the structure and the content of the documents to be generated; we discussed some of the novel features of our tool, such as element sharing, and described how ToXgene can be extended or coupled with other tools. Finally, we reported on preliminary experiments we conducted with our tool.

Development of ToXgene is continuing. We intend to provide a mechanism to allow the generation of text according to different vocabularies, grammars, and character encoding schemes. This would be of capital importance for generating testing data for text-intensive applications. With respect to further validating our tool, we plan to try to reproduce other benchmarking data sets, and by doing so, we expect to identify limitations and opportunities for improving our tool. We also intend to improve the performance of ToXgene; in particular, we are interested in exploiting possible parallelism in the data generation.

As future research, we identify the extraction of ToXgene templates from existing documents as an interesting problem. The obvious application of this research would be generating synthetic data closely reproducing the characteristics of real documents. This would be especially important for the cases where one has no access to the real data, say for security reasons. Also, templates capture considerable information about the data they describe, and thus are valuable metadata in themselves.

References

[1] A. Aboulnaga, J. F. Naughton, and C. Zhang. Generating synthetic complex-structured XML data. In *Proceedings of the Fourth International Workshop on the Web and Databases*, pages 79–84, Santa Barbara, CA, USA, May 24-25 2001.

[2] D. Barbosa, A. Barta, A. Mendelzon, G. Mihaila, F. Rizzolo, and P. Rodriguez-Gianolli. ToX - the Toronto XML engine. In *Proceedings of the International Workshop on Information Integration on the Web*, pages 66–73, Rio de Janeiro, Brazil, April 9-11 2001.

[3] P. Bohannon, J. Freire, P. Roy, and J. Siméon. From XML Schema to Relations: A Cost-based Approach to XML Storage. In *Proceedings of the 18th International Conference on Data Engineering (ICDE)*, 2002.

[4] T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler. Extensible markup language (XML) 1.0 (second edition) - W3C recommendation. Available at `http://www.w3.org/TR/2000/REC-xml-20001006`, October 6 2000.

[5] D. C. Fallside. XML Schema part 0: Primer - W3C candidate recommendation. Available from `http://www.w3.org/TR/xmlschema-0/`, October 24 2000.

[6] IBM DB2 Universal Database XML Extender - administration and programming. Available at `http://www-4.ibm.com/software/data/db2/ extenders/xmlext/`, 1999.

[7] IBM XML Generator. Available from `http://www.alphaworks.ibm.com/tech/xmlgenerator`.

[8] ToX*gene* - the ToX XML data generator. `http://www.cs.toronto.edu/tox/toxgene`, 2001.

[9] Transaction Processing Performance Council. *TPC Benchmark H - Decision Support*, 1999. Revision 1.3.0.

A Detailed experimental results

In this section we give more details on the results of preliminary experiments with ToXgene, conducted at the IBM Toronto Lab. We compare our tool with the data generators of both TPC-H [9] (`dbgen`) and Xmark (`xmlgen`). We decided to use TPC-H in our experiments because it is a widely used relational benchmark and because it defines many non-trivial integrity constraints over its database. Xmark was chosen because it was designed specifically for XML applications, and defines a complex document, with different levels of nesting, and thousands of references among its elements.

Query	sizes		ratio	
	ToXgene	`dbgen`	size	time
q14	1	1	1.0000	1.0366
q12	2	2	1.0000	1.0000
q3	10	10	1.0000	1.0972
q20	21	15	0.7143	0.9276
q2	44	54	1.2273	1.2665
q16	2762	2789	1.0098	1.2853

Table 1: Comparison between ToXgene and `dbgen`, using queries from the TPC-H workload.

Feature	**Target**	ToXgene	`dbgen`
average length of comments for suppliers	63	61	61
average length of supplier addresses	25	24	25
average balance on supplier accounts	4500	4450.81	4473.30
average balance on customer accounts	4500	4510.06	4470.50
average size of parts	25	25.00	25.00
average available quantity of parts	5000	5004.00	5000.00
average supply cost of parts	5000	499.74	499.69
average tax per item in each order	0.04	0.04	0.04

Table 2: Comparison between ToXgene and `dbgen`, using *ad-hoc* queries.

All experiments were run on a 4-way 500 Mhz Pentium III Xeon machine that has a total of 3Gb of RAM and 18Gb of disk storage.

Experimental methodology. For each benchmarking data, we run two queries: queries chosen from the benchmark workload, and *ad-hoc* queries. Our goal in executing the queries of the workload is to verify whether the data we generate offers the same "level of difficulty" as the data produced by the respective data generators. The *ad-hoc* queries are designed to verify whether the distributions of certain properties of the data are comparable.

We report the results of each experiment on a separate table. For each query, we report the size of its result on each data set, the relative size of the result sets (defined as $\text{size}(tg)/\text{size}(\texttt{gen})$, where *tg* is a value measured using the data produced by ToXgene, and `gen` is the value obtained using the other data generator), and the relative time to execute that query (computed similarly as the size ratio).

For the TPC-H experiment, we generated data for a 100Mb relational database (i.e., we use a scaling factor of 0.1), corresponding to over 400Mb of XML content, divided into 6 documents. The data generation process for TPC-H takes about 1 hour and uses no more than 750Mb of RAM for storing the lists. We loaded this data into an empty relational database in DB2 V7.1 using DB2's XML Extender [6]; the schema of this database was identical to the one populated by `dbgen`, thus the same integrity constraints are defined in both databases. The metric for answer sizes was the number of tuples in the result set of each query.

For Xmark, we generated a 100Mb document (i.e., we use a scaling factor of 1). On average, it takes about 25 minutes to generate such a document on the machine we used; 250Mb was the maximum amount of memory used. We used Kweelt[3] to execute both the queries from the Xmark workload and our *ad-hoc* queries. We specified all queries in a way such that each data value is output in a separate line in the answer document. Thus, we can use the number of lines in the answer as the metric for comparing the sizes of the results of the queries. We comment on these results below.

A.1 TPC-H results

The TPC-H database defines small (25 tuples) and large (150,000 tuples) tables; several integrity constraints are defined over these tables. The data generated by ToXgene was loaded successfully, which indicates that no

[3]Available at `http://http://kweelt.sourceforge.net/`.

Query	sizes		ratio	
	ToXgene	`xmlgen`	size	time
q11	7661	7661	1.0000	1.0086
q1	12	12	1.0000	1.0097
q20	25	25	1.0000	0.9736
q17	25595	25535	1.0023	1.0100
q2	35425	34841	1.0168	1.0200
q12	1358	1436	0.9457	0.9888
q14	236	173	1.3642	0.8689

Table 3: Comparison between ToXgene and `xmlgen`, using queries from the Xmark workload.

Feature	**Target**	ToXgene	`xmlgen`
average price in closed auctions	100.00	100.58	96.83
average number of bids per open auction	5	5.01	4.96
average "happiness" of customers with closed autions	5.50	5.48	5.43
percentage of items in Europe that have "United States" as location	0.75	0.7553	0.7447
percentage of items in Asia that can be purchased with credit card	0.50	0.5020	0.4945
percentage of persons that have a profile	0.50	0.5043	0.5029
average income of customers	42,500	42,543.68	42,604.98

Table 4: Comparison between ToXgene and `xmlgen`, using *ad-hoc* queries.

referential integrity constraint was violated. Table 1 shows the results of some queries from the TPC-H workload, the first three queries in the table return a fixed number of tuples. Table 2 shows the *ad-hoc* queries we ran. All random values in the TPC-H data set are generated using uniform probability distributions.

A.2 Xmark results

Xmark defines an auction database whose main objects are items, persons, categories, and auctions (both open and closed). There are thousands of references among elements in these categories (e.g., each auction corresponds to one particular item and has one person as the seller). We validated the document generated by ToXgene using the DTD provided with the benchmark without problems. Moreover, the time required for parsing both our document and the one generated by `xmlgen` was around 22 seconds.

Generating Xmark text. Most of the textual content in this benchmark is generated by sampling from 17000 most common words in the Shakespeare's plays, according to their frequency of occurrence in those documents. We obtain similar content by loading the list of words used by `xmlgen` into a list, which is sampled accordingly. Sentences are formed by concatenating these words. We generate the final text by copying these sentences into `emph`, `bold` and `keyword` elements in a recursive fashion, so that we can have `bold` sentences nested within `emph` sentences.

Table 3 shows the results of the queries selected from the Xmark workload. The top three queries have fixed output size. As shown in the table, the relative result sizes and execution times for the queries are very close, except for q14, which returns the number of occurrences of a given word inside elements having textual content.

Finally, Table 4 compares some properties of both documents w.r.t. the expected values as defined by the benchmark. The features presented in that table are determined by different probability distributions. For instance, the average prices of items in closed auctions obey an exponential distribution, while the number of items that can be bought with a credit card obey a uniform distribution.