XML Data Cleaning with XQuery and Extension Functions

Diogo Simões
INESC-ID and IST, Lisbon

Abstract

The eXtended Markup Language (XML) is currently the language of choice for storing and exchanging data across diverse application domains. Moreover, the XQuery language has emerged as a powerful and standardized way for querying XML documents. However, and despite its expressiveness, XQuery still lacks appropriate mechanisms for effectively handling XML data cleaning problems. Data cleaning refers to the process of correcting anomalies in a dataset. These anomalies may be for instance be due to typographical errors or duplicate representations of real world objects. With the growing amount of XML data, approaches to effectively and efficiently clean XML are clearly required, although this issue is not addressed by existing data cleaning systems, which are specialize on handling relational data. This paper advocates the usage of XQuery together with extension functions for XML data cleaning. I discuss the advantages of such an approach, introduce some example application scenarios, and detail the implementation on top of an existing XQuery engine. Experimental results compare the proposed approach with a typical ETL (Extract, Transform and load) tool and with an XML data cleaning prototype.

1 Introduction

Data cleaning aims at obtaining high quality data, by converting source into target data without errors, inconsistencies, nor duplicates. This activity is crucial in several application contexts, such as in data warehousing, data integration, and data migration.

In the last decade, effective and efficient relational data cleaning techniques have been exhaustively studied. However, substantially less effort has been put into the problem of XML data cleaning, a more challenging task since the structure and schemas of XML data tend to be more complex than their relational counterparts.

In this paper, I argue that a possible approach relates to equipping XQuery with a data cleaning extension, the W3C standard language for querying XML data, together with mechanisms to identify and correct data quality problems so that it is possible to clean effectively and efficiently any XML data source.

The rest of this paper is organized as follows: Section 2 introduces fundamental concepts related to XML data management. Section 3 presents related work on XML data cleaning. Section 4 suggest a taxonomy of data quality problems that may occur in XML databases. Section 5 presents a methodology for XML data cleaning. Section 6 details the proposed XQuery extension functions for XML data cleaning. Section 7 presents some initial validation experiments, in which I used an XML dataset to illustrate the usefullness of the proposed extension functions, and the feasibility of using XQuery for XML data cleaning. In Section 8, I compare the execution time required for using XQuery to write the extension functions, instead of resorting to other languages, likely more efficient. Finally, Section 9 presents our conclusions and points directions for future work.

2 XML Data and XML Query Languages

Nowadays, the eXtended Markup Language (XML) has become the language of choice for storing and exchanging data across diverse application domains. Moreover, the XQuery language introduced by the World Wide Web Consortium offers a powerful and standardized way to query XML-encapsulated information. With its ability to integrate XML and non-XML data, to perform up-

dates over XML repositories through the XQuery Update extension, or to query textual information through the XQuery Full-Text extension, XQuery is a nowadays a powerful tool for interacting with large repositories of XML data. However, XQuery still lacks appropriate mechanisms for efficiently supporting the definition of XML data cleaning programs. In terms of expressiveness, XQuery is Turing complete, as it does not restrict recursion in user-defined functions.

Similarly to SQL for the case of relational databases, XQuery contains functions for extracting, summarizing, aggregating, and joining data from potentially very large XML datasets. The language builds on XPath (i.e., another W3C standard for specifying navigation paths over XML documents) and on XML Schema's type system, combining elements from two domains, namely the presentation domain and a computational domain, into a single combined syntax. The result is that any atomic value or XML document (i.e., a particular data presentation) is already a valid XQuery expression, which evaluates to itself. There are also language statements, such as *for* and *let,* which can be intermixed with XML elements or XPath expressions.

Figure 1 illustrates an XQuery expression, providing the reader with a sense of XQuery's expressiveness and capabilities. The query illustrates most of XQuery's key features, namely i) path expressions for navigating, selecting, extracting and joining XML values, ii) constructors for creating new XML values, iii) *let* expressions for binding variables to intermediate results, iv) *for* expressions for iterating over sequences and for constructing new sequences, and v) functions for modularizing queries and performing elementary processing.

The current XQuery standard already has a substantial set of built-in predicates and functions that capture common tasks related to data aggregation, handling dates and numeric computations, or processing text. Moreover, standardized extensions such as XQuery Full-Text offer extra functionalities with regard to particular aspects (i.e., keyword search in the case of XQuery Full-Text). I propose to leverage on the expressiveness of XQuery, extending it with a function library that can facilitate the writing of data cleaning programs.

```
let $a := <dataset>
<prop_a id="id1" value="one value"/>
  </dataset>
let $b := <dataset>
<prop_b id="id1" val="another value"/>
  </dataset>

for $u in $a//prop_a[id="1"]
let $t := $u/@value
return
  <new_prop id="{$u/@id}">
    {for $v in $b//prop_b[@id = $u/@id] return
       <props>
          <prop_a>{upper-case($t)}</prop_a>
          <prop_b>{$v/@val}</prop_b>
       </props>
    }
</actors>
```

Figure 1: An example XQuery FLWOR expression.

3 Related Work on XML Data Cleaning

While research abounds in the realm of relational data cleaning, there is yet little work for data cleaning in other, more complex data models, such as XML. Nonetheless, some exceptions are worth mentioning.

Several authors have for instance addressed the specific XML data cleaning subtask of duplicate detection. An example is the work by Weis and Naumann, where the authors proposed a generalized framework for duplicate detection from which they derived an XML duplicate detection method named DogmatiX [11]. In brief, the general framework divides the duplicate detection problem into three components, namely (i) candidate definition to choose which objects are to be compared, (ii) duplicate to specify defining when two duplicate candidates are in fact duplicates, and (iii) duplicate detection specifying how to efficiently find those duplicates. DogmatiX specifically compares XML elements based not only on their direct data values, but also on the similarity of their parents, children, structure, etc. Weis and Naumann proposed heuristics to determine which of these to choose, as well as a similarity measure specifically geared towards the XML data model.

Aiming at improving the efectiveness of XML data duplicate detection, and noticing that the XML structure can indeed have a significant impact on the process of duplicate detection, Leitão et al. proposed a method

that better captures the likelihood having nodes and descendant nodes being duplicates, through the use of a probabilistic approach based on Bayesian networks [5]. In subsequent work, Leitão and Calado proposed a novel method that automatically restructures XML elements in order to take full advantage of the relations between their attributes [4]. This new structure reflects the relative importance of the attributes in the data and avoids the need to perform a manual selection.

Focusing specifically on the efficiency of an XML duplicate detection process, Puhlmann et al. [8] extended a classical approach to duplicate detection in relational data, namely the sorted neighborhood method, to cover nested XML elements. The sorted neighborhood method draws its efficiency from sliding a window over the dataset and comparing only records within that window. To compare objects, the authors propose to make use of XML parent and child relationships, applying the windowing technique in a bottom-up fashion for detecting duplicates at each level of the XML hierarchy.

Cecchin et al. proposed a model for XML data fusion, supporting the definition of data cleaning rules for solving value conflicts that may have been detected previously [2]. The rules resemble decisions that are made by users when data are manually curated and, once defined, conflicts detected in subsequent integration processes that are within the context of existing rules can be automatically solved without user intervention. The authors also introduced a notion of fusion policy validation that prevents conflicting resolution rules to be defined. A prototype system named XFusion was developed to validate the proposed model, consisting of a rule-based cleaning tool that stores curated data in a integrated repository.

The work that is perhaps more similar to ours is that of Weis and Ioana, where the authors proposed an XML data cleaning framework named XClean, based on a set of cleaning operators whose semantics is well-defined in terms of XML algebraic operators [10]. In XClean, users specify data cleaning programs by combining operators through a declarative language named XClean/PL, which is then compiled into XQuery. The authors described XClean's operators, the XClean/PL language, and the general compilation approach, validating the effectiveness of the proposed framework through a series of case studies. In this paper, I instead argue that the XQuery language itself, enriched with appropriate functions, can be used directly to write the data cleaning programs.

4 A Taxonomy of Data Quality Problems

In the past few years, several classifications for the data quality problems that may occur in relational databases have been proposed. The goal of these data quality taxonomies is to understand which problems may occur in a database. Detach the Müller and Freytag [6], Kim et al. [3], Barateiro and Galhardas [1], Oliveira et al. [7] and by Rahm and Do [9] taxonomies for relational data cleaning.

The proposed taxonomy, represented in Table 1 distinguish two types of fata quality problems: those occurring in XML documents with a single schema and those which occurring in XML documents with multiple schemes. These two kinds of problems are in turn divided between the value level problems, element level problems and attribute level problems. The data quality problems can be also divided between the problems avoided by XSD and the problems not avoided by XSD.

4.1 Problems Occurred in XML Documents with a Single Schema (I)

I.1 Value-level problems are related with the value of elements and attributes of the XML documents. These problems can be further divided into problems that are avoided with a XML schema definition (XSD) and problems that a well defined XML schema cannot avoid.

I.1.1 Problems avoided by XSD: I consider the following ones:

- **Format Representation:** when two files have different format representations for the same information (e.g., the currency format can be represented by 3$50 or $3.50). It is possible to avoid this problem by defining a regular expression for the value in XSD.

- **Different Measurement Units:** when the same object is represented in different units (e.g., in a file the price is given in dollars and in another file,

Data Quality Problems		Avoided by XSD (x.y.1)		Not Avoided by XSD (x.y.2)	
		XML Docs w/ a Single XSD (I)	XML Docs w/ Multiple XSD (II)	XML Docs w/ a Single XSD (I)	XML Docs w/ Multiple XSD (II)
Value (x.1)	Format Representation	✓	✓	-	-
	Different Measurement Units	✓	✓	-	-
	Illegal Value	✓	✓	-	-
	Entity Integrity Violation	✓	✓	-	-
	Misspelling	-	-	✓	✓
	Erroneous Data	-	-	✓	✓
	Embedded Values	-	-	✓	✓
	Misfielded Values	-	-	✓	✓
	Ambiguous Data	-	-	✓	✓
	Missing Data in a non-null Field	-	-	✓	✓
Element (x.2)	Invalid Number of Elements	✓	✓	-	-
	Missing Data	✓	✓	-	-
	Different Element Structures	✗	✓	-	-
	Name Conflict	✗	✓	-	-
	Contradictory Elements	-	-	✓	✓
	Duplicate Elements	-	-	✓	✓
	Element Dependency Violation	-	-	✓	✓
Attribute (x.3)	Missing Data	✓	✓	-	-
	Name Conflict	✗	✓	-	-
	Attribute Dependency Violation	-	-	✓	✓

Table 1: A Taxonomy for data quality problems.

```
<document>
  <student id = "12345">
    <name>Homer J. Simpson</name>
    <course>MEIC-T</course>
  </student>
  <student id = "12345">
    <name>Peter Griffin</name>
    <course>MEIC-T</course>
  </student>
</document>
```

Figure 2: An example of Entity Integrity Violation.

the price is given in euros). It is possible to avoid this problem by defining a regular expression for the value in XSD.

- **Illegal Value:** when a value is outside of the domain range (e.g., birthdayDate = 1989-13-32). This problem can be avoided by defining the domain range of the value or its data type in XSD.

- **Entity Integrity Violation:** when a value that should be the key is associated to more than one element. In the example illustrated in Figure 2, the id attribute value must be unique.

I.1.2 Problems not avoided by XSD : this taxonomy considers:

- **Misspellings:** when a word is misspelled (e.g., "Petre Grifin" instead of "Peter Griffin").

- **Erroneous Data:** when the data is valid but does not conform to the real entity (e.g., a student has 20 years old but the value that appears is 30).

- **Embedded Values:** when different types of data are embedded in a unique value (e.g., "student Homer Simpson").

- **Misfielded Values:** when data is stored in the wrong field (e.g., country = "Los Angeles").

- **Ambiguous Data:** when data can be interpreted in various ways. This situation can occur when there is abbreviated data (e.g., Homer J. Simpson could be a reference to Homer Jeffrey Simpson or Homer Jay Simpson).

- **Missing data in a non-null field:** when an element or attribute is filled without any meaning (e.g., age = ?????).

```
<document>
  <student id = "12345">
    <name>Homer J. Simpson</name>
    <adress>Av. Liberdade n.2, Lisbon</adress>
  </student>
  ...
  <student id = "12355">
    <name>Homer Jay Simpson</name>
    <address>Avenida da Liberdade n2, Lisbon</address>
  </student>
</document>
```

Figure 3: An example of two duplicate elements in an XML file.

```
<document>
  ...
  <student name="J. Doe">
    <age>22</age>
    <bdate>21-12-1986</bdate>
  </student>
  ...
</document>
```

Figure 4: An example of element dependency violation.

I.2 Element-level problems are the problems that can occur in the elements of an XML document. These are as follows:

I.2.1 Problems Avoided by XSD

- **Invalid Number of Elements:** when the number of elements of a certain type is not valid (e.g., the same person having more than one identity number).
- **Missing Data:** when an element that must have a value does not have it.

I.2.2 Problems Not Avoided by XSD

- **Contradictory Elements:** when there are two different elements that represent the same object in real world which have contradictory information.
- **Duplicate Elements:** when there exist two different elements that represent the same element in the real world. Figure 3 exemplifies this data quality problem.
- **Element Dependency Violation:** when two or more elements that are related do not satisfy this relationship. Figure 4 presents an example where the age does not match the birthday value.

```
<document>
  <student name="J. Doe" age="22" bdate="21-12-1986"/>
  ...
</document>
```

Figure 5: An example of attribute dependency violation.

I.3 Attribute-level problems are the problems related with the attributes of an XML document. These can be:

I.3.1 Problems Avoided by XSD

- **Missing Data:** when an attribute that must have a value, does not have.

I.3.2 Problems Not Avoided by XSD

- **Attribute dependency violation:** when two or more attributes that are somehow related do not satisfy this relationship. In the example of Figure 5, the age does not match the birthday date.

4.2 Problems Occurred in XML Documents with a Multiple Schema (II)

The data quality problems that may exist in a single source may persist when considering multiple sources are integrated. In this section, we present in detail only the problems that can occur in multiple XML files.

The **II.1 Value-level Problems** which occur in XML documents with multiple schemes are exactly the same of those which occur in XML documents with a single schema.

In addition to the **element-level problems (II.2)** which occur in XML documents with a single schema, other problems may occur when I am dealing with XML documents with multiple schemes (both **Avoided by XSD (II.2.1)**):

- **Name Conflict:** when different objects are represented with the same tag name (synonymous) or the same object is represented with different tag names (homonyms).
- **Different Element Structures:** when an object is represented in different ways in different sources. The examples illustrated in Figure 6 and Figure 7 represent two XML documents referring the same object but represented differently.

```
<document>
  <student id = "12345">
    <name>
      <firstname>Homer</firstname>
      <givennames>J. Simpson</givennames>
    </name>
  </student>
</document>
```

Figure 6: document1.xml presents a structure for student element which separates the first name and last name in different elements.

```
<document>
  <student id = "12345">
    <name>Homer J. Simpson</name>
  </student>
</document>
```

Figure 7: Differently for document1.xml the student element of document2.xml represents the student full name as a unique element.

As for the **attribute level problems occurring in multiple XML documents (II.3)**, they are the same that may occur in a single document, plus the **name conflict** problem that occur when different objects are represented with the same attribute name (synonymous) or the same object is represented with different attribute names (homonyms).

5 A Methodology for XML Data Cleaning

Relatively to the process of XML Data Cleaning, I argue that is possible to create a methodology which ensures the cleaning of XML data sources.

In relational databases it is possible to ensure the data cleaning of relations ensuring the cleaning of their tuples, which in turn are cleaned if the problems of their attributes are resolved. Since the XML structure varies depending on the data source, the methodology for XML data cleaning must be different and has to take into account the hierarchical structure (tree based) of XML data. Whence, considering the hierarchical structure of XML, in which elements may have other descendant elements, it is possible to ensure the cleaning of the top elements if their descendants are cleaned.

It is thus possible ensure the cleaning of a XML data source with a recursive bottom-up process where, from the descendants to the ascendant nodes, until the parent node, the tasks of data correction, normalization and duplicate elimination are performed over the elements.

6 XQuery Extension Functions for Data Cleaning

I argue that the following groups of data cleaning functions are fundamental: (i) formatting and normalization; (ii) string matching; (iii) dictionary-based matching; (iv) consolidation; and (v) conversion. I summarize the semantics of each group in the remaining of this section.

6.1 Formatting and normalization

These functions return a normalized value for a given field, depending on its type. I separate this class of functions into two sub-groups: those that handle basic types (such as date, time, etc), and those that handle complex entities, such as addresses, phone numbers, city and country names, and zip codes.

6.2 String matching

String matching functions are used to compare a pair of strings and return their similarity or distance. The following types of string matching functions are supplied:

- Sequence-based: Edit distance, Needleman-Wunch, Jaro, Jaro-Winkler.
- Set-based: Overlap, Jaccard, TF/IDF.
- Hybrid: Generalized Jaccard, Soft TF/IDF.
- Phonetic: Soundex, Metaphone.

6.3 Dictionary-based Matching

These functions return the most similar entry in a dictionary for a given input word. The following functionalities were supplied:

- Spellings: Given a word (potentially in any language), check the spelling, and return the right spelling in case of error.
- Synonyms/antonyms Given a word, return its synonym/antonym.

- Abbreviations: Given a word, returns its abbreviation or expansion.

6.4 Consolidation

Consolidation functions are essentially user-defined aggregation functions. They are applied to a set of values of the same type and return a single value. They are useful when we need to merge or consolidate data records. The following list gives some examples of functions were supplied:

- Most frequent: returns the element of the set that is the most frequent.
- Longest/shortest: returns the element of the set that is the longest (string), or biggest (other data type).
- Super: if all the elements in the set are substrings of one of them, return this one.

6.5 Conversion

Conversion functions are used when we need to convert measurement units. The following ones are provided:

- Kg/Pounds
- Km/Miles
- Farneheit/Celcius

7 Experimental Validation

In order to illustrate the usefulness of the proposed extension functions, and the feasibility of using XQuery for XML data cleaning, I wrote some simple data cleaning programs (using three different approaches) to clean an XML dataset describing information about music artists and CD albuns, which had been previously used in the context of XML duplicate detection experiments. The dataset includes 5000 of the 9763 CDs existing in CDDB, a dataset randomly extracted from freeDB.

A characterization of the CDDB[1] dataset is given in Table 2, showing for each of its XML elements the number of corresponding instances, the number of distinct instances, and an example value (the most frequent value of each element).

```
<cddb>
  <disc>
   <artist>Shane Barnard</artist>
   <dtitle>Psalms</dtitle>
   <category>folk</category>
  </disc>
  <disc>
   <artist>Shane Barnard</artist>
   <dtitle>Psalms</dtitle>
   <category>misc</category>
  </disc>
</cddb>
```

Figure 8: Example of duplicate elements in the CDDB dataset.

In terms of data quality problems, and specifically in what regards duplicates, the CDDB dataset is distributed together with information regarding the approximately duplicate CD elements existing in the data (i.e., a total of 298 CDs correspond to duplicates of other CDs described in the dataset). I used XQuery together with the proposed extensions in order to write a simple data profiling program, from which the data quality problems described in Table 3 could also be infered.

Again using XQuery together with the proposed extensions, I wrote a program to fix the existing data quality problems, following the typical approach of decomposing the data quality process into a sequence of steps, which covers (i) data normalization, (ii) duplicate detection, and (iii) duplicate resolution.

With the XQuery program illustrated in Figure 9, from these 5000 CDs, just 54 pairs of duplicates was misidentified.

In order to compare the extension functions for XQuery with other approaches, I implemented two other programs. One program was implemented with the Microsoft SSIS software[2], the other was implemented with the XClean prototype [10]. So it was possible to verify that SSIS is the efficient way to clean XML data - the dataset with 5000 tuples was cleaned in 106 seconds whereas XQuery took 44679.8 seconds to clean this dataset. However, XQuery is the more effective approach: from the 5000 records, just 54 pairs was misidentified against the 104 misidentified pairs of SSIS program.

[1] http://www.hpi.uni-potsdam.de/naumann/projekte/repeatability/datasets/cd_datasets.html

[2] http://msdn.microsoft.com/en-us/library/ms141026.aspx

XML node	Occurrences	Distinct Occurrences	Most Frequent Value
/cddb	1	1	-
/cddb/disc	9763	9763	-
/cddb/disc/did	9763	9763	7f0980a,a809780a
/cddb/disc/artist	9974	7562	Various
/cddb/disc/dtitle	9963	9659	Greatest Hits
/cddb/disc/category	9763	11	misc
/cddb/disc/cdextra	4064	2599	YEAR: 2003
/cddb/disc/year	5219	76	2002
/cddb/disc/genre	6393	593	Rock
/cddb/disc/tracks	9763	9763	-
/cddb/disc/tracks/title	63402	57896	Intro

Table 2: Statistical analysis of CDDB nodes - Number of occurrences, number of distinct occurrences and the most frequent value of each node.

XML Node	**Problem Category**	**Problem**	**Example**
/cddb	-	-	-
/cddb/disc	Element	Duplicate Data	See Figure 8
/cddb/disc/did	-	-	-
/cddb/disc/artist	Value	Misspelling Ambiguous Data	Ilse DeLangue Mozart, W.A.
	Element	Duplicate Data	Mozart, W.A. / W.A. Mozart
/cddb/disc/dtitle	Value	Erro Misspelling Format Representation	Hibrid Theory Contrasts / contrasts
/cddb/disc/category	-	-	-
/cddb/disc/cdextra	Value	Misspelling Format Representation	nEclipse YEAR 2004 / Year; 2004
/cddb/disc/year	Value	Missing Data in a non-null field Erroneous Data Format Representation	9999 2070 60 / 1960
/cddb/disc/category	Value	Format Representation	hiphop / Hip Hop
/cddb/disc/tracks	-	-	-
/cddb/disc/tracks/title	Value	Embedded Values Missing Data in a non-null field Format Representation	Track 1 - Welcome / Welcome ? 1-Best Of You Best Of You

Table 3: Data quality problems identified in the CDDB dataset.

XClean needs more memory to perform the data cleaning program, not being able to clean a dataset with 5000 tuples. Therefore, to compare XQuery with XClean the dataset was reduced to 300 tuples. XQuery was found to be more efficient than XClean but equally effective. The XQuery program cleaned this dataset in 358.5 seconds and XClean cleaned these 300 CDs records in 389 seconds. Both misidentified one pair of duplicate CDs.

8 Cost of Portability

Instead of implementing the data cleaning library of functions in XQuery, it would be possible to resort to another programing language such as Java or C++, which are likely more efficient. However, I choose to write these functions existing in XQuery, allowing its portability and making it possible to use them in any XQuery processor. To verify what is the execution time cost of this choice, a comparison was made between the execution time of several string matching functions written in XQuery and Java.

To compare XQuery with Java, I selected 8 string

```
let $data := doc("cddb.xml")
let $no-misspellings :=
 spell:correction ($data,
     ("/disc/artist", "/disc/dtitle"),
     ('hibrid theory', 'va', 'various', 'v.a', 'v.a.'),
     ('Hybrid Theory', 'Various Artists',
      'Various Artists', 'Various Artists',
      'Various Artists'))
let $normalized-data :=
  norm:normalization ($no-misspellings)
let $no-art-dup-data :=
  dup-elimination:dup-elimination($normalized-data,
        0.8, dup-elimination:artist-dups-elimination#2)
let $no-dups :=
dup-elimination:dup-elimination ($no-art-dup-data,
          0.8, dup-elimination:cd-dup-elimination#2)
return $no-dups
```

Figure 9: XQuery program for cleaning CDDB dataset.

matching functions existing in our XQuery library (see Section 5). The functions selected include the four types of string matching functions: Sequence-based, set-based, hybrid and phonetic-based. The sequence-based functions used was edit-distance, Jaro-Winkler and Needleman-Wunch. To test the set-based functions, I chose Jaccard-tokens, Dice-tokens and Cosine-tokens (the tokens considered in these functions were the words that compose the strings). The hybrid and phonetic-based functions chosen were, respectively, Soft-Cosine-Jaro-Winkler and Soundex.

In this experiment, we used strings regarding north American national parks from the *Match*[3] dataset. This dataset has 250 strings, composed with a maximum of 71 characters and a maximum of 14 words. Figures 10(a) and 10(b) describe the distribution of the strings size in number of characters and number of words, respectively.

With this experiment was possible to conclude that Java is much more efficient than XQuery. In all the functions tested, the implementation in Java had better results than the XQuery approach, especially the Sequence-based and the hybrid functions. Set-based and phonetic-based functions had results more identical. Still, even in the best case for XQuery, Java is 3 times faster. Figure 11, demonstrates these results.

[3] `http://www.cs.cmu.edu/wcohen/data`

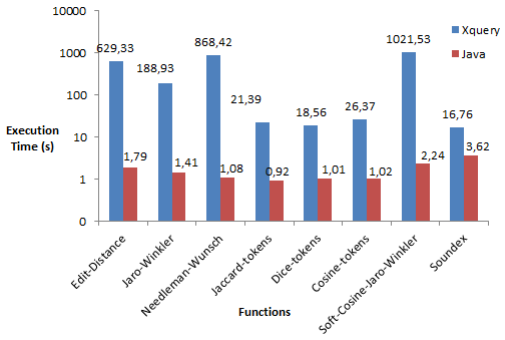


Figure 11: Execution time of the several similarity functions.

9 Conclusions and Future Work

The results presented in this work demonstrate that XQuery with the extension functions described in Section 6 is a reliable approach to perform data cleaning in XML databases. However as it is described in Section 8, XQuery still have a performance problem to be solved. XQuery execution times are incomparably higher than other languages.

Placed this, I assume that the approach presented in this work could be a better alternative to perform XML data cleaning with some improvements, in the future. The main challenge open for future work is the optimization of the XQuery functions in order to make this a reliable alternative to other existing approaches to perform data cleaning. Another work that could be done in the future, to improve the results generated by the data cleaning programs implemented with the data cleaning library, is the support for user feedback and the interactive cleaning.

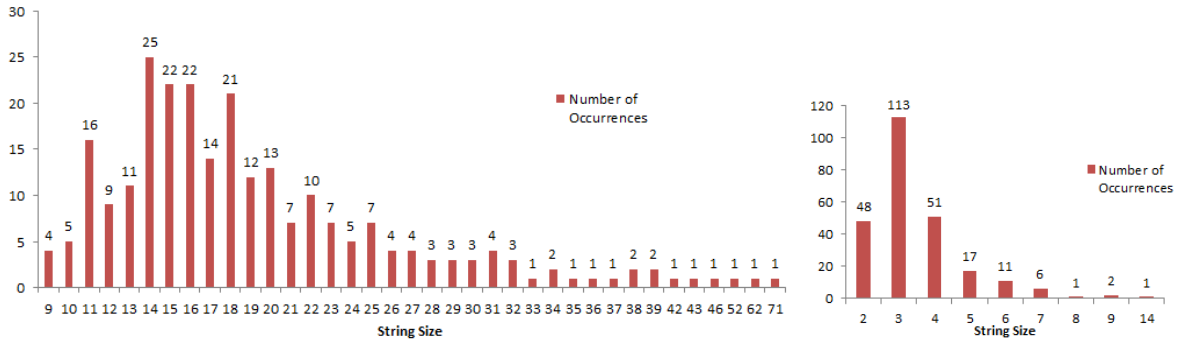
References

[1] J. Barateiro and H. Galhardas. A survey of data quality tools. *Datenbank-Spektrum*, 14:15–21, 2005.

[2] F. Cecchin, C. D. de Aguiar Ciferri, and C. S. Hara. XML Data Fusion. In *DaWak*, pages 297–308, 2010.

[3] W. Kim, B.-J. Choi, E.-K. Hong, S.-K. Kim, and D. Lee. A taxonomy of dirty data. *Data Mining Knowledge Discovery*, 7:81–99, January 2003.

[4] L. Leitão and P. Calado. Duplicate detection through structure optimization. In *CIKM*, pages 443–452, 2011.



((a)) ((b))

Figure 10: Distribution of the strings size in: (a) number of characters; and (b) number of words.

[5] L. Leitão, P. Calado, and M. Weis. Structure-based inference of XML similarity for fuzzy duplicate detection. In *CIKM*, pages 293–302, 2007.

[6] H. Müller and J. Freytag. *Problems, methods, and challenges in comprehensive data cleansing.* Informatik-Berichte // Institut für Informatik, Humboldt Universität zu Berlin. Inst. für Informatik, 2005.

[7] P. Oliveira, F. Rodrigues, P. R. Henriques, and H. Galhardas. A Taxonomy of Data Quality Problems. In *2nd Int. Workshop on Data and Information Quality*, pages 219–233, Porto, Portugal, June 2005. (in conjunction with CAiSE'05 conference).

[8] S. Puhlmann, M. Weis, and F. Naumann. XML Duplicate Detection Using Sorted Neighborhoods. In *EDBT*, pages 773–791, 2006.

[9] E. Rahm and H. H. Do. Data Cleaning: Problems and Current Approaches. *IEEE Data Engineering Bulletin*, 23:2000, 2000.

[10] M. Weis and I. Manolescu. Declarative XML data cleaning with XClean. In *Proceedings of the 19th international conference on Advanced information systems engineering*, CAiSE'07, pages 96–110, Berlin, Heidelberg, 2007. Springer-Verlag.

[11] M. Weis and F. Naumann. DogmatiX Tracks down Duplicates in XML. In F. Özcan, editor, *ACM SIGMOD Int. Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*, pages 431–442. ACM, 2005.