Data Cleaning Model for XML Dataset using Conditional Dependencies

Mohammed Hakawati, Yasmin Yacob, Rafikha Aliana A. Raof, Jabiry M. Mohammed, and Eiad Syaf Alhudiani

Abstract—Data Cleaning - as an essential phase to enhance the overall quality used for decades with different data models, the majority handled the relational dataset as the most dominant data model. However, the XML data model also considered within the most data models commonly used for storing, retrieving, and querying valuable data. In this paper, we introduce a model for detecting and repairing XML data inconsistencies using a set of conditional dependencies. Detecting inconsistencies will be done by joining the existed data source with a set of patterns tableaus as conditional dependencies and then update these values to match the proper patterns using a set of SQL statements. This research considered the final phase for a full cleaning model introduced for XML datasets by firstly mapping the XML document to a set of related tables then discovering a set of conditional dependencies (Functional and Inclusions) and finally applying the cleaning algorithms as a final step of quality enhancement.

Index Terms—XML, Data Quality, Conditional Integrity Constraints, SQL.

I. INTRODUCTION

Data is now becoming the lifeblood for companies as various database systems, such as Decision Support Systems, Customer Relationship Management, Big Data industry Projects, and Internet of things systems, are being used; valuable information and expertise can be obtained from a large amount of data. Nevertheless, investigations indicate that many of such applications are not running successfully and efficiently due to a number of issues, such as poor system development or poor database reliability, but nothing is sure to cause application failure than lack of data quality issues [1].

According to studies and reports published by V12-Data in 2015, the cost of bad data could be considerably higher than 12% of lost revenue. Approximately 28% of individuals who had issues relating to the delivery of e-mails indicated that customer service had endured accordingly, while 21% suffered damage to credibility. The vast majority of organizations (86%) acknowledged that their data might be wrong in some way, whereas, 44% of businesses and organizations reported missing or incomplete data as the most frequent issue alongside obsolete information [2].

The impact of poor data quality can be of three types:

1. Operational (causing customer and employee dissatisfaction and increased costs),
2. Tactical (affecting decision making and causing mistrust),
3. Strategic Impacts (affecting the overall organization’s strategy).

Overall, any system or enterprise that heavily relies on data is prone to experience problems if the data being handled does not possess the expected data quality [3].

Extensible Markup Language (XML) stands out rapidly amongst essential data file formats. It has been used for scientific data such as DNA sequences, to annotate extensive documents such as DrugBank database, or for exchanging data over the Web for e-commerce benefits.

Authors in [4] provide in-depth analysis to answer the question “Is the quality of XML documents found on the web sufficient to apply XML technologies like XQuery, XPath, and XSLT?” The results show that on the web, 58% of the existing documents over the web are of XML file format, nevertheless, one-third of these documents accompanying valid XML Schema Definition (XSD) or Document Type Definition (DTD). Moreover, about 14% of the documents lack well-formedness. A simple error of mismatching or missing tags will render the entire XML technologies useless over these documents.

The growing interest of XML as the dominant way of exchanging data over the Web encourages researchers to address XML data cleaning as an open research problem [5], and to start searching for data cleaning approaches for XML, especially approaches based on Integrity Constraints (IC) [6]–[8].

II. LITERATURE REVIEW

Data cleaning is a crucial process for organizations that seek to extract valuable information from raw data. The majority of XML cleaning solutions can be classified along two lines; Duplicate Detection and Data Integration. Indeed, they are two vital research topics for web technologies. Duplicate or record linkage focuses on finding the same entities referred to like the same real-world objects, whereas, integration is essential for new database applications to ease the merging of data documents from many resources and reconciles between their schema and instance.

On the contrary, a few numbers of XML cleaning
approaches based on IC have been presented for many reasons. Firstly, earlier integrity constraints provided approaches for enhancing schema (DTD or XSD) as an improving XML document’s quality rather than enhancing data instances. Moreover, the notations of ICs in XML are not unified like relational databases; some notations depend on the concept of the path whereas the others depend on the tree tuples concept.

The first real moves toward cleaning the XML tree [9], [10] was proposed a technique with minimum modifications to turn XML data into a consistent situation and prepare it to answer user’s queries. The repair phases described involved changing the value of an attribute or the content of an element, or by marking some of the attributes or elements of T as unreliable. The following definition describes the repairing techniques adopted to produce a consistent tree using XFD.

**Definition 1:** Reliability of XML Tree is a RX = (T, ϕ) Where T is an XML tree, and ϕ is a reliability function from a set of nodes (N(T)) to {true, false}, such that for each pair of nodes n₁, n₂ ∈ N(T), if n₂ descendant of n₁, it holds that (n₁) = false ⇒ (n₂) = false. To be able to create a repair, the RX tree must not satisfy XFD according to the definition of weak satisfiability.

**Definition 2:** Being given an XML tree T conforming to a DTD and a FD f: S₁ → S₂, XML tree T satisfies f (T ⊨ f), if for each pair of tree tuples t₁, t₂ ∈ T, if t₁, S₁ = t₂, S₁ ∧ t₁, S₂ ≠ ∅ then t₁, S₁ = t₂, S₂. By given a set of FDs = {f₁, ..., fₙ} over D, then (T ⊨ FDs) if (T ⊨ fᵢ), ∀ i ∈ [1, n].

The main issue with this approach is that the complexity time was undesirable to decide whether there exists a repair for an XML document in the presence of a DTD and FDs.

Improving XML data quality with FD [11] was the first demonstrated experimental approach applied to highlight the importance of XML ICs in data quality, by using a cost-based model to modify violated data, and provide an efficient two-step heuristic method for repairing XFD violations. The advantage of this approach (XRepair) is the complexity time; by defining upper and lower bounds for optimum repair, their algorithm is still in NP-complete in the size of the data even though when using fixed DTD and set of FDs. The main shortcoming for the XRepair is that the approaches need a weight for each node inside the XML documents, the truth is assigning weights needs user interaction, whereas assigning the weights randomly - as proposed - will not give confidence about the accuracy and consistency for repaired documents.

The aforementioned cleaning approaches do not consider user interaction in the cleaning process, even though the perspective of the user as the definer of ICs is important. Incorporated user interactions during the cleaning phase was a work done in [6], in addition, to introduce the concept of repair groups. The user can play a role in assigning nodes weights and choosing the preferred repairing method (Modifying node values or make them unreliable), also, the user has the ability to choose the repair needed candidate.

The advantage of this approach is making a rapier even with few modifications and overcome the limitations of the approach introduced by [9], [10]. Unlike in the original approach, this approach does not choose the candidate repair that modifies the value of the nodes; but choosing the repair that marks a node as unreliable by adding a coefficient k to the calculation of the repair candidate cost.

Previously mentioned XML cleaning techniques can be classified into two directions: modifying inconsistencies by changing node values or by marking inconsistent nodes as unreliable. Moreover, all these techniques hired XFD as a single type of XML ICs to detect and correct data inconsistencies. Finally, the majority of these techniques suffering from high complexity time.

Previous techniques have assumed that dependencies are already known as either being extracted from business rules or designed by domain experts [6], [12]–[14] which is not always right. Practically, it is necessary for useful dependency-based cleaning methods to have in place a technique that can automatically discover or learn required dependencies from the XML data tree T to be used as data cleaning rules. Indeed, it is often unrealistic to solely rely on human experts to design XCFDs and XCINDs via an expensive and long manual process. As indicated by Gartner [15], “cleaning rules discovery is critical to commercial data quality tools”.

Many lessons can be learned from relational databases admire that some inconsistencies can be found from different tree tuples; unfortunately, XFD can only cope with inconsistencies under the same tree tuple. Furthermore, considering a set of XML ICs assigned by users and then checking their consistency with the existed dataset is a time-consuming phase; as after a long process of chasing, the set of ICs may contradict with each other and with dataset consistency. The limitation of traditional XML dependencies and the informality between their notations and the availability of many XML schemas played an important role in making old data cleaning approaches inadequate and inefficient in capturing data inconsistencies.

**III. METHODOLOGY**

The proposed model contains two main steps after XML Initialization as a mapping XML to a set of tables and stores them in a database [16] as appear in Fig 1.
The first step involves mining data quality dependencies (XCFD, XCIND) that should be followed by the data in order to considered valid. However, these phases already introduced in previous research [17], [18]. While the second step involves the use of these dependencies with existing data tables to find another consistent dataset that minimally differs from the original by clean data inconsistencies. Therefore, a minimal, nontrivial set of pattern tableaus for discovered conditional dependencies will be stored in a separate relation and used for the cleaning phase.

A. Pattern Tableaus

Table 1 illustrates the scheme for the patterns table that holds all information about all discovered (or even assigned) XML condition dependencies (XCFD, XCIND). To show how this summary table is formulated, the requirement of metadata for both dependencies and the pattern values needed will be explained.

For each XCFD 𝜙 as a dependency connects columns in a single relation; the lhs Selector is the table where this dependency holds and the lhs, rhs conditional columns are metadata and lhs, rhs values are the pattern data values. On the other side, for each XCIND 𝜙, lhs, rhs selectors are tables where the dependency holds, lhs, rhs fields are used to connect tables, whereas lhs, rhs conditional columns are considered XCIND metadata. lhs and rhs values as pattern data values.

The pattern table can help reduce the time and complexity needed to join participant relations for each side of the dependency with a related pattern tableau to discover inconsistencies. The proposed algorithm for improving XML quality will read the required data only for the summary table and generate a set of denial queries. If no result is sent back from these queries, then no inconsistent data violates related dependency.

B. Detecting and Repairing Inconsistencies

A data value can be considered inconsistent in a situation where it violates an explicit declaration (like a master database) of its correct value. Given an XML Tree T (a set of tables stored in a database) and a set Σ = (𝜙 | 𝜙) of conditional dependencies over T, in order to test the consistency, the algorithm computes its denial query, extracting those tuples that do not satisfy the constraint in negated form. Consistency holds if the denial query returns an empty result. [19], [20].

For functional dependencies (XFD, XCFD) case, a path instance violates a specific data dependency if it matches lhs of the dependency, but do not match the rhs of the dependency [21]. In more details, as shown in the definition of conditional dependencies in [17], a tree tuple values are considered inconsistent if one of the following situations appear:

T violate an XCFD: 𝜙 = 𝑝 → {𝑒}, {𝑋} → {𝑌}, denoted by T ≠ 𝜙 , iff
1) For each path 𝑝 ∈ Paths(T) and a single pattern tableau 𝑡𝑝 ∈ T𝑝 , 𝑝(𝑙ℎ) ≠ 𝑡𝑝(𝑙ℎ), but 𝑝(𝑟ℎ) ≠ 𝑝(𝑟ℎ).
2) For each two paths 𝑝 , 𝑝’ ∈ Paths(T) and a single pattern tableau 𝑡𝑝 ∈ T𝑝 , 𝑝(𝑙ℎ) ≠ 𝑝’(𝑙ℎ), but 𝑝(𝑟ℎ) ≠ 𝑝(𝑟ℎ).

<table>
<thead>
<tr>
<th>Constraint Type</th>
<th>lhs Selector</th>
<th>lhs Fields</th>
<th>Lhs Cond</th>
<th>lhsCond Values</th>
<th>rhs Selector</th>
<th>rhs Fields</th>
<th>rhsCond Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>XCFD</td>
<td>Book</td>
<td>Publisher</td>
<td>'Berkeley'</td>
<td></td>
<td>Genre</td>
<td>'Science'</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Order</td>
<td>Title, Author</td>
<td>Type</td>
<td>'Book'</td>
<td>Book</td>
<td>Title, Author</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Order</td>
<td>Title, Author</td>
<td>Type</td>
<td>'Article'</td>
<td>Article</td>
<td>Title, Author</td>
<td></td>
</tr>
</tbody>
</table>
The first situation concerns about constant XCFD whereas the second one concerns about variable XCFD. For instance, in the following example, the mining algorithm discovered a constant XCFD, \( \varphi: ([\text{Author}, \text{Genre}] \rightarrow [\text{Publisher}], \text{tp}) \), with a tableau in (Table 2). The violations appear when there exists a book with the Author’s name is ‘Ray Bradbury’ and the book Genre is ‘Science,’ but the publisher is not ‘Diogenes Verlag.’

On the other side, Inclusion Dependencies (XIND, XCIND) provide different criteria for classifying a data values as inconsistent [18]; any value that appears in the dependent subtree (\( \text{lhs} \)) should also appear in the referenced subtree (\( \text{rhs} \)). This valued fact puts us at crossroads, which paths should be modified to make inclusion dependency satisfied. For example, in the order subtree, there exist a book discovered using a generated query based on \( \text{XCIND } \psi:\{\text{order } [\text{Title}, \text{Author}; \text{Type}] \subseteq \text{book } [\text{Title}, \text{Author}, \text{tp}] \}

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
\textbf{Author} & \textbf{Genre} & \textbf{Publisher} \\
\hline
‘Ray Bradbury’ & ‘Science’ & ‘Diogenes Verlag’ \\
\hline
\end{tabular}
\end{table}

The truth is modifying the referenced subtree (book) asks to insert a new path into the \( \text{rhs} \) subtree to make dependency satisfied. On the other side, modifying dependent subtree (order) needs to delete \( \text{rhs} \) subtree to make dependency satisfied. Authors in [22] presented a chase algorithm for inserting new path values to make them obey related dependency. Authors in [22] presented a chase algorithm for inserting new path values to make them obey related dependency. Authors in [22] presented a chase algorithm for inserting new path values to make them obey related dependency. Authors in [22] presented a chase algorithm for inserting new path values to make them obey related dependency. Authors in [22] presented a chase algorithm for inserting new path values to make them obey related dependency.

This solution looks more effective because deleting \( \text{lhs} \) values will cause new anomalies. Furthermore, inclusion dependencies asked \( \text{lhs} \) values to be available in the \( \text{rhs} \) values regardless of \( \text{rhs} \) size.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
\textbf{Order} & \textbf{Book} \\
\hline
\text{Title} & \text{Author} & \text{Type} & \text{Title} & \text{Author} \\
\hline
‘ipv6 essentials’ & ‘Silvia Hagen’ & ‘book’ \\
\hline
\end{tabular}
\end{table}

\section*{Algorithm 1: XML Cleaner (\( \Sigma, T \))}

\begin{algorithm}
\begin{algorithmic}[1]
\Require XML document as a set of \( R_p, \Sigma = \{ \varphi \cup \psi \} \)
\Ensure Modified XML Tree \( T' \) \[ T' = T \]
\State \textit{vio }\{\}; //store inconsistent values 
\State \textbf{for each} \( \varphi: (\text{lhs } \rightarrow \text{rhs}; \text{tp}) \in \Sigma \)
\State \quad \textbf{for each} \( R_p \mid (\text{Attr}(R_p) \cap \varphi (\text{lhs}, \text{rhs}) \neq \emptyset) \)
\State \quad \textit{vio }\{\text{SELECT DISTINCT } * \text{ FROM } R_p, \varphi, t_p \}
\State \quad \text{WHERE } R_p \cdot \text{lhs }\neq t_p \cdot \text{lhs}
\State \quad \text{AND } R_p \cdot \text{rhs }\neq t_p \cdot \text{rhs}
\State \textbf{for each} \( T' = \text{Repair} (\varphi, \text{vio}, T) \)
\State \textbf{for each} \( \psi: (\text{lhs } \subseteq \text{rhs }; \text{tp }) \in \Sigma \)
\State \quad \textbf{for each} \( R_p \mid (\text{Attr}(R_p) \cap \psi (\text{lhs}) \neq \emptyset) \)
\State \quad \textbf{for each} \( R'_p \mid (\text{Attr}(R'_p) \cap \psi (\text{rhs}) \neq \emptyset) \)
\State \quad \textit{vio }\{\text{SELECT DISTINCT } \text{lhs }\text{ FROM } R_p, \text{tp } \}
\State \quad \text{WHERE } \text{lhs }\text{ NOT IN}
\State \quad \text{SELECT } \text{lhs }\text{ FROM } R_p, R'_p, \text{tp }
\State \quad \text{WHERE } R_p \cdot p \neq R'_p \cdot p
\State \quad \text{AND } R_p \cdot \text{lhs }= t_p \cdot \text{lhs}
\State \quad \And R_p \cdot \text{lhs }= t_p \cdot \text{lhs}
\State \textbf{for each} \( \text{Repair} (\psi, \text{vio}, T) \)
\State \textbf{return} \( T' \)
\end{algorithmic}
\end{algorithm}

In XML Cleaner algorithm, lines (1-5) checked each XCFD against related \( \text{lhs} \) Selector \( S (R_p \text{ table}) \), if the result \( \text{vio} \) is empty, then no violated data discovered, otherwise these values modified to proper patterns values by invoking Repair procedure. Lines (7-15) checked each XCIND against related \( \text{lhs} \text{Selector, rhsSelector subtrees, if there exists a tree tuple in lhs of the dependency and not at rhs, then it needs an insertion as a requirement of the solution.}

The repair procedure starts by identifying the type of dependency so as to decide the corresponding repair. If an XCFD has inconsistent value, then the Repair procedure will update the \( \text{rhs} \) conditional paths of the dependency to match \( \text{rhs} \) of pattern tableau. otherwise, if an XCIND has inconsistent value, the procedure will insert all \( \text{lhs} \) Fields elements of the dependency into the related \( \text{rhs} \) subtree if it has satisfied the proper condition (\( \text{lhs} \) conditional paths).
IV. IMPLEMENTATION

For testing, prototyping the proposed cleaning model, a piece of application programmed to verify and validate. Microsoft SQL Server 2016 was used to store and retrieve data and querying for rules. Furthermore, C# programming language was used to develop the cleaning application GUI and pass parameters in and out SQL Server. Moreover, LINQ (Language INtegrated Query) component was implemented to add native data querying capabilities. XML datasets were imported into the SQL server using the XML Reader, which is a reader that provides fast, non-cached, forward-only access to XML data. XML Schema (XSD) was generated from XML tree using Visual Studio, stored and used for XML shredding and reconstructing XML tree for data repairing and cleaning.

The repairs committed by the XML Cleaner satisfy all dependencies $\Sigma = \{\varphi \cup \psi\}$. Nevertheless, in some cases, a new error may appear during the repairing process because of cyclic dependencies. To measure the quality of repair, the notions of Recall and Precision are adopted as they represent completeness and correctness respectively. The precision is computed as the ratio of the number of correctly repaired node values to the number of nodes detected values, whereas the Recall is the ratio of the number of correctly repaired nodes to the number of total error nodes.

$$\text{Precision} = \frac{\text{Number of repaired data nodes}}{\text{Number of detected data nodes}}$$

$$\text{Recall} = \frac{\text{Number of repaired data nodes}}{\text{Number of noised data nodes}}$$

To prove the efficiency of the proposed cleaning technique; the number of cleaned inconsistencies should be as near as possible to the noise ratio. Therefore, in this research, the value of noise ratio will vary from $\mu = 15\% - 35\%$ of the XML tree analogously with XRepair (Tan & Zhang, 2011b). To add noise to the XML tree, the value of existing values is changed to a new value randomly. More precisely, data values of the elements on the rhs of the XCFD are changed from the correct to incorrect values, whereas new tree tuples are added to the lhs of XCIND and obey a particular condition. Furthermore, 10 XFD used for FDRrepairer and XRepair, whereas all mined XCFDs used for XML Cleaner.

Figure 2 shows the result of the Precision test by comparing XML Cleaner with XRepair and FDRrepairer. As appear, XML Cleaner can enhance the quality of XML document over XRepair and FDRrepairer with about 94% of the average. The point is XML Cleaner matches all leaf paths with discovered patterns and any data values do not match the pattern considered inconsistent. Moreover, XML Cleaner able to repair more detected inconsistencies, more precisely, any approach that able to repair a high number of inconsistencies and do not cause a new violation, then this approach has a higher precision value for the quality issue.

![Fig 2](image-url)

**Fig 2**: Accuracy of XML Cleaner, XRepair, and FDRrepairer (Precision).

On another side, Figure 3 illustrates the same comparison using the Recall test. The results show that XML Cleaner is always outperformed XRepair and FDRrepairer with more than 83% of the average. The values of Recall are obtained by comparing the ratio between the repaired value and inserted dirty values. Therefore, XML Cleaner can modify values with almost all inserted inconsistencies for XCFD and XCIND. Finally, the scalability of XML Cleaner is not similar to XRepair and FDRrepairer regarding data repair, XML Cleaner algorithms behave descending according to the noise ratio

![Fig 3](image-url)

**Fig 3**: Accuracy of XML Cleaner, XRepair, and FDRrepairer (Recall).

V. CONCLUSION

The main aim of this study was enhancing the XML dataset quality by introducing a driven data cleaning model based on new derived types of data integrity called Conditional Functional Dependencies (XCFD) and Conditional Inclusion Dependencies (XCIND). Regarding the running time of discovering patterns, the scalability shows that the time elapsed is almost constant and depends on the number of tree
tuples (Table Tuples). XML Cleaner proves its efficacy to improve the accuracy of the XML dataset by enhancing the correctness and completeness of the dataset regarding a discovered set of conditional dependencies.

The results showed that the new model could increase XML quality by detecting more real spurious data values than previous models based on traditional dependencies. Furthermore, the XML Cleaner can sense inconsistencies between the same tree tuples or even between multilevel tree tuples inside the XML tree using the mentioned conditional dependencies. Moreover, the quality of the documents was assessed using two measures (Precision and Recall), and the accuracy of XML documents was improved over 94%, 83% respectively for these measures.

Where no cyclic dependencies mined during patterns discovery, XML Cleaner outperforms other approaches, which are based on single dependency type (XFD) and traditional full dependencies, by enhancing the quality of the XML tree with a higher Precision and Recall values. Similar to relational databases; XML conditional dependencies are able to play an important role in enhancing the consistency and to optimize the overall trustworthiness of available datasets over traditional full dependencies.

ACKNOWLEDGMENT

This research was partially supported by Cetic research center, we thank our colleagues from the DSIDE department in Cetic. We also thank our colleagues from the University of Malaysia Perlis who provided insight and expertise that greatly assisted the research. Furthermore, we would like to thank Eiad Saif Alhudiani for his technical and programming support and for providing ideas to enhance the vision of the work.

REFERENCES