Journal of Theoretical and Applied Information Technology
15th September 2017. Vol.95. No.17

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

CONDITIONAL INCLUSION DEPENDENCIES FOR IMPROVING XML DATA CONSISTENCY

[1]MOHAMMED HAKAWATI, [1]YASMIN YACOB, [1]RAFIKHA ALIANA A. RAOF, [1]AMIZA AMIR, [1]JABIRY M.MOHAMMED [2]EYAD SAIF AL-HODIANI

[1]School of Computer and Communication Engineering
University Malaysia Perlis, Campus Pauh Putra, 02600 Arau, Perlis, Malaysia

[2]Amman Arab University, Amman, Jordan

E-mail: [1]mshakawati@hotmail.com.

ABSTRACT

Without any doubt, XML data model considered the most dominant document type over the web with more than 60% of the total; nevertheless, their quality is not as expected. XML integrity constraint just as its relational counterpart played an important role to keep XML dataset as consistent as possible, but their ability to solve data quality issues is still intangible. The main reason is old-fashioned data dependencies introduced mainly to keep schema consistent rather than data consistent. In this paper, a conditional version of XML inclusion dependencies (**XCIND**) is proposed for data quality issues and justify the ability to use inclusion dependencies for data quality issues. **XCIND** Notations will extend **XIND** and shift its mission from schema design to data quality by providing pattern tableaus. Moreover, a set of minimal **XCIND** dependencies will be discovered and learned using a set of mining algorithms. Finally, the ability to use **XCIND** to detect data inconsistencies will be inspected using denial quires between mined rules and XML tree.

Keywords: *XML, Data Quality, Data Cleaning, Integrity Constraints.*

1. INTRODUCTION

Today, data become the lifeblood of business, with the varied use of database applications, like Decision Support Systems, Customer Relationship Management Systems, Data Warehouses, Web Services, and eLearning Systems. Beneficial information and knowledge can be gained from considerable amounts of data using these applications. Nevertheless, investigation shows that lots of such applications fail to run successfully and effectively, there are many reasons to cause the failure, such as poor system design or query performance, but nothing is more certain to yield failure than lack of concern for the issue of data quality [1].

According to a study presented from Data Mentor's blog in 2015, the expense of bad data (with all data models) may be even higher than that 12% lost revenue. 28% of those who have had problems delivering email say that customer service has suffered as a result, while 21% experienced reputation damage. Most of the companies, 86%, admitted that their data might be inaccurate in some way. Whereas 44% of businesses said, missing or imperfect data are the most frequent problem with outdated information [2].

With the increasing significance of XML as the main data model for data transfer and data integration, data quality becomes a critical issue to make these applications success. Data cleaning, which refers to a set of processes used to improve data quality, has been used extensively in relational databases with less concern in XML [3].

The need to effectively manage business information, which is filled with inconsistencies, incompleteness and stored in XML documents, is currently more important than any time. numerous investigations by specialists underline the value of effective and efficient techniques for handling “erroneous data" at scale [4]. In spite of the fact that this issue has gotten critical consideration over time in the traditional database literature, XML cleaning approaches fall far short of an effective solution for big data and web data [5]. Cleaning XML databases pose new challenges and problems not faced in cleaning relational databases, the first challenge is the semi-structure tree model which more difficult to

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

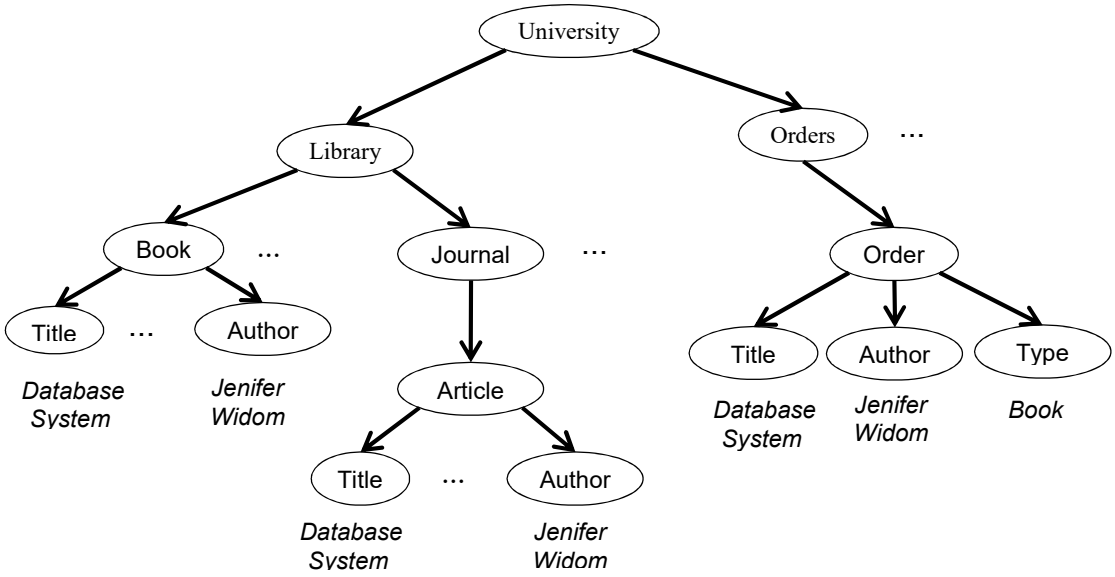

Figure 1: Sample XML data Tree.

handle than relational one, the second challenge is that there are no unified notations for XML integrity constraints [6].

In the database theory, good data quality can be evaluated using five main attributes (Accuracy, Currency, Deduplication, Completeness, and Consistency), each attribute employed to solve a specific data quality problem [7]. Data consistency concerns about making the dataset obey a set of rules defined by expert domains, acquired using crowdsourcing or mined from existing dataset using a set of data mining algorithms.

Traditionally, in order to guarantee schema consistency, integrity constraints are essential [8]. Functional and inclusion dependencies are the most widely recognized integrity constraints used for schema design issues. Nowadays, data consistency is also important for all data models (structured and semi-structured).Figure 1: Sample XML data Tree.

Conditional dependencies have been introduced lately for the relational data model to analyze and improve data quality. As their main role is enhancing the quality of a single table (CFD) [9] or between pairs of relations (CIND) they only cover a small portion of the data instance that matched a specific condition [10].

XML Conditional Functional Dependencies (XCFD) [11], presented recently as a first step to fill the gap between XML Functional Dependencies (XFD) and data quality problem, Nevertheless, inclusion dependencies are also needed to find inconsistencies between multi-hierarchal levels inside the same XML tree [12].

Figure 1 shows an XML dataset contains information about a university library with a finite number of books and journal articles to be used for online borrowing system and transferring transactions during daily orders. The system will use information stored in the XML file to check available resources (Books, Articles) and return order's query results. A set Σ of two traditional XML Inclusion Dependencies (XIND) hold at the XML tree T and used for schema matching are defined as follows:

$$\psi_1: ((uni/orders/order\ [Title, Author]) \subseteq (uni/library/book\ [Title, Author]))$$

$$\psi_2: ((uni/orders/order\ [Title, Author]) \subseteq uni/library/journal/article[Title, Author]$$

First XIND insists that if an order requested by a student, it should find a matching entity in the book subtree in library books, completely as the second XIND requests that the order finds a match in available articles. These dependencies are satisfied by given XML document tree ($\Sigma \vDash T$) and required full agreement between both sides of the dependency. Nevertheless, these two inclusion dependencies do not make sense; how can the same order match elements from two different subtrees at the same time? Now, suppose a student asked for an article titled like “Database” for author “Jenifer Widom," the system will check the first XIND and found that the requested order satisfies the first XIND, so the query result will back a book, which contains the same requested Title and Author, But, with a wrong type. To solve this problem and *w.l.o.g.*

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Let us add a conditional constraint element to *lhs* of the dependencies to become:

$$\psi_3: (uni/orders/order\ [Title, Author\ ; Type = 'book']) \subseteq (\ uni/library/book\ [Title, Author])$$

$$\psi_4: (uni/orders/order[Title, Author; Type = 'article']) \subseteq (uni/library/journal/article[Title, Author])$$

XIND_3, ψ_3 asserts that for each order in the orders subtree, if the element Type = 'Book', then there must exist a text node in the book subtree *s.t.* $order[Title, Author] = book[Title, Author]$)

Moreover, XIND_4, ψ_4 asserts that for each order in the orders subtree, if its type element ='Article', then there must exist a text node in the articles subtree such that $(order[Title, Author] = article[Title, Author])$.

These enhanced dependencies are XIND that holds only on a subset of the XML document which satisfies the condition rather than the entire document, in addition, to be checked only using the related dependency, not BOTH.

In light of these, there has been increasing demand for data quality tools, for effectively detecting errors in the XML data. Here, we introduce a new class of XML dependencies especial for detecting data inconsistencies between multilevel of the tree.

This paper organized as follows: Section 2 provides basic preliminaries for XIND, patterns tableaus, and other definitions. Section 3 covers the notation and related issues for proposed XML conditional inclusion dependencies (XCIND). Section 4 presents a set of algorithms to mine a set of approximate XCIND with enhanced data structures and to derive a minimal set of patterns as an initial step toward a new era of XML data cleaning models. Section 5 provides an algorithm to discover XML inconsistent values. Section 6 argues about our result of mining dependencies and tools adopted, and finally, Section 7 concludes and outlines future work.

2. PRELIMINARIES

Integrity constraints can be defined as a set of properties or rules that should be satisfied by every data instance (XML data path) of a database [8]. Data which violate these constraints considered as inconsistent data and requires cleaning. Since there is more than one way of making these changes, methods have been proposed to clean data using different ways such as cost and distance functions [13][14][15], The goal of such approaches is to make the cleaned database as close as possible to original one.

Obviously, XML integrity constraints have attracted much interest in last two decades to ensure data consistency [16]. The expression "integrity constraint" in XML used to mean extensions of relational integrity constraints, such as, functional, inclusion, approximate dependencies and so on, which depend principally on the equality of data values within a single or multi-relations.

Even though XFD played a tiny role in instance cleaning, XIND had nothing to remember. Bohannon et al. [13] recommended that not only functional dependencies but also inclusion dependencies required for data cleaning as well as schema design. This motivates database theory scientist to introduce a massive number of approaches for improving relational database using both types' of dependencies.

2.1 XML Inclusion Dependencies

Inclusion dependencies are the type of constraints which connect two set of attributes, between two different or even same relations, for instance, $R_1[A, B] \subseteq R_2[A', B']$ means that all values of the dependent attribute A, B of R_1, are contained in the value set of the referenced attribute A', B' of R_2 [8]. XML inclusion dependencies are important in many fields like XML publishing, where the relational database has to map to a single predefined XML Schema [17].

Karlinger et al. [18] presented XIND as a dependency preservation when mapping relational database to XML dataset, they used the same notation offered by their old work for XFDs [19], the most notable difference in their notations and others notations is they don't use any XML Schemas (DTD or XSD) but using the closest node.

Their notations for XIND has a form of $((P, [P_1, \dots, P_n]) \subseteq (P', [\ P_1', \dots, P_n']),$ where P and P' paths are *lhs* and *rhs* selectors respectively, and $[P_1, \dots, P_n]$ and $[\ P_1', \dots, P_n']$ are a nonempty set of paths called left hand side , right hand side (*lhs*, *rhs)* fields respectively. This notation differ slightly from the previous XML inclusion notation in that it considers only simple paths for both selectors and field paths, in addition, to allow attribute and text nodes rather than element node types.

Many other notations for XIND presented [17], we cannot mention all of them for lack of space. However, XIND does not present any effort toward data cleaning and data quality, even though they played an important role in improving schema and instance in the relational database.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

2.2 Conditional Constraints

Fan et al. [9] developed a new extension to traditional dependencies for improving data instance, the idea was the first big step toward new comprehensive data cleaning approaches. Hundreds of papers published to study how CFD improve data quality [20][21] and it's siblings like data mining and data integration.

XML database is not less important of relational one as mentioned earlier, this motivation leads authors in [11] to conduct a conditional copy of XFD called XCFD, and modify XFD notation based on generalized tree tuple (GTT) of to convey the new rules. XCFD Notation is an expression of the form $pv: \{e\}, \{X\} \rightarrow \{Y\}$, where: v is the context path, $\{e\}$ is the conditional part, and $\{X\} \rightarrow \{Y\}$ is a standard XFD. The conformation between the tree and the XCFD. $T \vDash \varphi$ achieved if two paths agree conditionally on their *lhs* then their *rhs* should match as well, this type of functional dependency is important for data cleaning issues rather than schema design issues and holds on a subset rather than entire documents

GTT [22] and XCFD mining algorithms adopted *partitions* concept for discovering XFD and its conditional version respectively, the idea of partitions presented mainly in [23] as one of the most effective ways to deal with large databases with more care about relation degree (number of columns in a relation). To use partitions manner the authors used XML representation to shredding the XML document into a set or relations using pivot node concept which in turn gives semi-normalized table using XSD schema.

2.3 XML Data Representation

XML tree can be seen, as a set of tuples inside relational tables, the idea behind tree tuple is to find a relative representation of XML tree as a relational data model. Some of the key important points about the mapping phase are that tuples hold related information can easily be access with different DBMSs and gives a clearer conceptual view for XML dataset [22], [24]. Moreover, discovered patterns take tableau form, so it becomes much easier to handle XML tree as tabular form and combine them at advanced levels.

Mapping XML tree can be done in many ways, Firstly by shredding the whole tree into a single huge flat relation, this concept has many disadvantages like the unmatched number of unreal tuples and a large size of degree, therefore, a direct result of this mapping is the increased complexity time [25].

On the other hand, another way is using a semi-normalized form by shredding XML into many related flat relations using the concept of tuple class. This representation uses the concept of schema normalization and reduced time and space.

University: Rcd
  Library: Rcd
    Book: SetOf Rcd
      Title: str
      Author: str
      ...
    Journal: SetOf Rcd
      Article: SetOf Rcd
        Title: str
        Author: str
      ...
Order: SetOf Rcd
    Title: str
    Author: str

Figure 2: University XML Schema.

2.3.1 Essential Tuple Class

The tuple in relational databases defined as a set of related data belongs to the same entity, projecting this idea to XML has numerous benefits and usage. In our case, tuple concepts are important to retrieve all data for a single entity and matching it with other tuples in order to discover data inconsistencies (future step).

GTT [22] notation presented as an enhanced version over XML tree tuple [25]. GTT granted that related data under the same Pivot Path could not be lost and support the concept of a set of elements. A tuple class C_P of the data tree T, contains all related GTTs, t_n, where path p_n called the *Pivot* Path and the last node of the path called pivot *node n*.

Each *repeatable* path (see [22] for more information) which contains maxOccurs (SetOf) restriction within the XSD schema (Fig.2) is taken into consideration. Each essential tuple class C_P corresponds to a unique set of schema element. Each created temporary relation (R_P) related to a specific essential tuple class C_P, by implementing the concept of essential tuple class, each non-repeatable element within an XML tree considered an attribute within related relation R_P, moreover each general tuple considered a relational tuple.

Table 1 shows three main relations scheme gained from mapping XML doc (and Journals hid as not needed) using class tuple Pivot Path with two additional columns Key and parent Kay as

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

preordered traversing number (hidden in the lake of space).

Table 1: Main Relational Schemas for University XML Dataset.

$R_{Article}$				
DOI	*Author*	*Title*	*Year*	*Journal*

R_{Order}			
OID	*Title*	*Author*	*Type*

R_{Book}					
ISBN	*Title*	*Author*	*Year*	*Publisher*	*Genre*

2.4 Pattern Tableaus

Pattern tableau can be defined as a subset of tree tuples in which the underlying conditional integrity constraints hold. A pattern tableau T_P restricts tree tuples of *lhs* subtree over attributes X_P (conditional context) and tree tuples of *rhs* subtree over attributes Y_P. For each repeatable path [A] (due XML Schema, XSD) in X_P or Y_P and each tuple $t_p \in T_P$, t_p[A] is either a constant in dom (A) or a special value '–'.

Each pattern tuple t_p defines a condition which is a constant value, t_p[A] restricts a matching tuple's attribute value to a constant value $'a' \in dom$ (A), and dash '–' represents an arbitrary data value from dom (A). A tree tuple $t_1 \in T$ matches $t_p \in T_P$, $(t_1 \asymp t_p)$ if $\forall A \in (X_P \text{ or } Y_P)$: $t_p[A] = ('-' \vee t_1[A])$ the pattern tableau is divided into *lhs* (with attributes X_P) and *rhs* (with attributes Y_P) as depicted in Table 2. Both sides of the tableau can be left empty specifying no restriction on any attribute of the respective relation. Below are the pattern tabulate for ψ_3 and ψ_4 respectively:

Table 2: Patterns Tableaus for ψ_3 and ψ_4.

T_1	Type	Nil
	Book	-

T_2	Type	Nil
	Article	-

3. XML CONDITIONAL INCLUSION DEPENDENCIES

This section introduces the syntax and semantics of the new XML dependency type, which is called XCIND and see the most important measures related to any dependency language; satisfiability, implication.

3.1 Syntax

XCIND syntax is too close to strong path notations; as this notation is the closest notation to relational notation and the clearest description of XML inclusion dependencies. XCIND, ψ is a pair:

$$\psi : \Big(\big((S\,,[P\,;\,P_c]) \subseteq (S',[P'\,;\,P_c'\,])\big), T_P\Big)$$

Where: $\big((S\,,[P]) \subseteq (S',[P'])\big)$ is a standard XML inclusion dependency [18] embedded in ψ such that: $S\,, S'$ are two paths called *lhs* ,*rhs* selectors respectively, $[P], [P']$ are two nonempty sequence of paths called *lhs* ,*rhs* fields respectively defined over XML Tree. $\forall i \in [1,n]$, $S.P_i$ and $S'.P_i'$ are legal paths end with attribute or text. $[P_c]$ *and* $[P_c']$ are two nonempty sequence of paths ends with text match values from patterns tableaus and called *Conditional Paths*. T_P is a pattern tableau, it contains all disjoint path values $(S.P_{ci})$ and $(S'.P_{ci}')$. For each repeatable path [A] in previous paths and for each $t_p \in T_P$, t_p[A] is a constant 'a' from Dom(A).

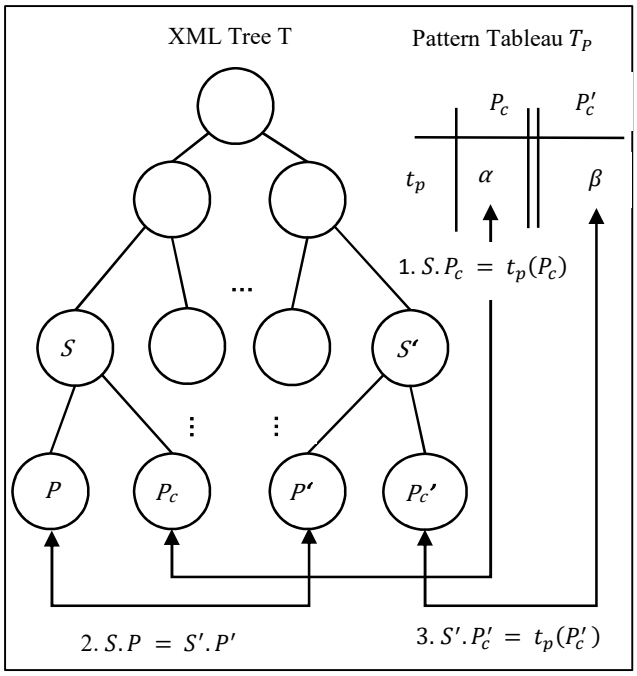

Figure 3: Tree, Tableau Confirmation.

3.2 Semantics

Suppose an XCIND ψ, which contains a traditional XIND as an embedded dependency with associated pattern tableau T_P. the embedded XIND semantics can be verified using *closest* node definition presented in [18], closest node property demands that all *lhs* selector and fields should belong

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

to the same path tuple and the same thing for *rhs*. Moreover, each *lhs* path should find a match *rhs* path. As a result, all *lhs* paths can be seen as tuples in a relation related to other tuples in different relation as *rhs* paths. Moreover, pivot node property presented by [22] request that related values for a dependency should belongs to the same GTT tuple class, both notations can help us in defining the semantics of XCIND and its relationship with XCFD.

A condition over the *dependent subtree* S should distinguish tuple paths P of S that included in the *referenced subtree* S' from tuples not included. A condition filtering only included tuples with a valid condition. The degree of validity (*Support*) can be regarded as the "precision" of a condition. Furthermore, a condition should filter all included tuples; its degree can be regarded as the *confidence* of a condition. Practically embedded XIND cannot completely cover *S* subtree; it applies only to *S* paths that matching certain pattern tuples t_P. More specifically, paths within XCIND matches the pattern tableau if $S.P_{ci} = t_p(P_{ci})$ and $S'.P'_{ci} = t_p(P'_{ci})$.

XML tree T satisfies XCIND ψ, denoted by ($T \vDash \psi$), iff for each conditional path $P_{c1} \in lhs$, and for each pattern tuple $t_p \in T_P$; if the value of the path P_{c1} *matches* the value of the corresponding tableau t_p ; $S.P_{c1} = t_p(P_{c1})$, then there exists a path $P'_2 \in rhs$, which match the non-conditional path of P_1; $S.P_1 = S'.P'_2$ and moreover its corresponding conditional path P'_{c2} *matches* the value of the pattern tableau; $S'.P'_{c2} = t_p(P'_{c2})$ (see Fig.3).

The satisfaction of the traditional XIND can appear clearly from above; when the value of the *lhs* path equal the value of the *rhs*, and this proof us that the embedded XIND still valid within the new notation. Intuitively, XML tree T satisfies a set Σ of XCINDs denoted by ($T \vDash \Sigma$), if ($T \vDash \psi$) for each $\psi \in \Sigma$. Moreover, two sets Σ_1 and Σ_2 of XCINDs are equivalent, denoted by $\Sigma_1 \equiv \Sigma_2$, if XML tree satisfies both, $T \vDash \Sigma_1$ and $T \vDash \Sigma_2$.

Pattern tableaus contain either value from the domain or special case '$-$' as a non-care value, logically, if a specific tableau t_P holds '$-$' for all its paths, then XCIND ψ covers the whole document and at that time there is no condition, which mean it's clearly XIND. This led us to think about XIND as a special case of XCIND. Finally, XCIND ψ: $\left(\left((S,[P;P_c]) \subseteq (S',[P';P'_c])\right), t_P\right)$ is in a normal form if T_P only consists of a *single* pattern tuple t_p. A set Σ of XCIND of the normal form can model an aggregated XCIND.

$$\Sigma(\psi): \left(\left((S,[P;P_c]) \subseteq (S',[P';P'_c])\right), t_p \in T_P\right).$$

3.3 Reasoning about XCIND

To make effective use of XCIND it is often necessary to reason about them. Two main issues related to any constraint language with any data model, the consistency problem (satisfiability) and the implication problem. Consistency insists that there is no contradiction between discovered dependencies, whereas implication helps to mine other dependencies using a set of inference rules [26].

3.3.1 The Satisfiability Issue

The satisfiability (consistency) problem of XCIND can be explained as follow: Given a set Σ of XCIND rules over an XML document tree T, expressional syntax ($T \vDash \Sigma$). The answer to the question "Is there exists a nonempty set of paths (data values) that conform a set of XCIND Σ ?" is significant.

In the presence of constraints, an XML document may result to be inconsistent if it does not respect some constraint. Assigning a set of traditional XIND Σ during the creation of the document require the satisfaction between them, but as known traditional dependencies are always satisfiable for relational [8] or even XML [19]. At the same line, inserting pattern tableaus for traditional XIND will not make it non-satisfiable, in contrast, the satisfiability problem is still achievable.

Theorem: XCIND is always, satisfiable.

proofing this theorem can be explained with two different techniques: Firstly: for relational constraints; a set CIND, Σ are always satisfiable by constructing a nonempty instance I such that $I \vDash \Sigma$, building the instance starts by looking for the *active domain* for each attribute inside each relation as an *lhs* in the database schema and then propagate these values to related attribute at the *rhs*. Next, using the patterns to expand the equality between non-conditional attribute for both *lhs* and *rhs* [10]. Section 4.3 introduces a set of algorithms to ensure the satisfiability of proposed XCIND by discovering a set of minimum dependencies from existing XML tree. As mining algorithms able to discover satisfied dependencies then $T \vDash$ XCIND.

Moreover, XML tree conforms a set of legal paths P if every path $p \in P$ ends with value v [18]. For two subtrees rooted by r_1, r_2 and has a set of paths under r_1 and r_2 , XML tree considered *complete* if all paths at *lhs* subtree r_1 ends with a set of values that are also have a match values at the *rhs* subtree r_2. Otherwise the XML tree considered to *confirms* paths P but not complete.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

3.3.2 The Implication Issue

XCIND Inference system is integrating of XIND and pattern tableau notation presented for a relational database. Below some of the main inference rules that can be used to discover a new dependency from the already discovered set and moreover reduce the number of discovered tableaus for the issue of reducing the processes needed to be done. These inference rules consider XCIND in the normal form and each tabulates contains a single conditional path. (note, $\vdash$ means implies).

IR_1: Reflexivity

$$: \Sigma \vdash \psi: \Big(\big((S, [P\ ;\ P_c]) \subseteq (S, [P\ ;\ P_c])\big), t_P\Big).$$

Where P is a sequence of distinct paths inside T and $t_P[p] = '-'$ for all $p \in P$.

IR_2: Permutation Projection

$$\psi: \Big(\big((S, [P_1 \dots P_m;\ P_c]) \subseteq (S', [\,P'_1 \dots P'_m;\ P'_c\,])\big), t_P\Big) \vdash$$
$$\psi: \Big(\big((S, [P_{\pi(1)} \dots P_{\pi(k)}; \hat{P}_c]) \subseteq (S', [P'_{\pi(1)} \dots P'_{\pi(k)}; \hat{P}'_c\,])\big), \hat{t}_P\Big)$$

Where $\{\pi(1) \dots \pi(k)\} \in \{\ 1 \dots m\}$, $\hat{P}_c$ and $\hat{P}'_c$ are a permutations for P_c , P'_c respectively, and $\hat{t}_P = t_P\,[P_{\pi(1)} \dots P_{\pi(k)};\ \hat{P}_c \parallel P'_{\pi(1)} \dots P'_{\pi(k)};\ \hat{P}'_c\,]$.

IR_3:Transitivity:

$$\psi: \Big(\big((S, [P; P_c]) \subseteq (\bar{S}, [\bar{P}; \bar{P}_c])\big), t_1\Big) \wedge \psi: \Big(\big((\bar{S}, [\bar{P}; \bar{P}_c]) \subseteq (S', [P'; P'_c\,])\big), t_2\Big) \vdash \psi: \Big(\big((S, [P; P_c]) \subseteq (S', [P'; P'_c\,])\big), t_3\Big).$$

Where $t_3[P\ ;\ P_c] = t_1[P\ ;\ P_c]$ and $t_3[P'; P'_c\,] = t_2[P'; P'_c\,]$.

Inference rules $(IR_1 - IR_3)$ are common for all data dependencies and can be found in all database textbooks [8], in this research we cover only these dependencies as they effected by pattern tableaus. Other XML rules like (Downshift, Upshift) are used only for displacement paths between selector and filed section, whereas Union rule used for combining n-ary inclusion dependency from a set on unary rules.

IR_4:Union:

$$\psi: \Big(\big((S, [P_1 \dots P_m;\ P_c]) \subseteq (S', [\,P'_1 \dots P'_m;\ P'_c\,])\big), t_P\Big) \wedge$$
$$\psi: \Big(\big((S, [P_{m+1} \dots P_n; P_c]) \subseteq (S', [\,P'_{m+1} \dots P'_n; P'_c])\big), t_P\Big)$$
$$\vdash \psi: \Big(\big((S, [P_1 \dots n;\ P_c]) \subseteq (S', [\,P'_1 \dots P'_n;\ P'_c\,])\big), t_P\Big)$$

Union rule IR_4 used to merge only rules shared same patterns and path selectors to imply new dependency rich with information. Actually, such inference rule is important to overcome XCIND inconsistences by allowing to add more paths values to rhs subtree as a solution for XCIND violations.

4. INFERENCING XCIND

XCIND Inference issue can be expressed as follows: Given an XML tree T over XSD S, find a cover set Σ of all XML conditional inclusion dependencies $\psi: \Big(\big((S, [P; P_c]) \subseteq (S', [P'; P'_c\,])\big), \mathrm{T_P}\Big)$ such that $\mathrm{T} \models \psi$ for all $\psi \in \Sigma$.

In order to discover a set of XCIND, the mining algorithm takes as input an XML document T as a set of related relations $\mathrm{R_P}$, and discover initially a set of *approximate* XINDs with support and confidence thresholds (θ, δ), then produced a set of minimal, non-trivial conditional inclusion. The reason of discovering traditional *approximate* XIND before start searching of pattern tableaus is that any attribute of $\mathrm{R_P}$ (repeatable path) cannot participate in both types of paths (conditional path and field path).

A set of $\mathrm{R_P}$ relations imposed to a set of preliminary phases to prepare them for discovering *approximate* XIND dependencies. As known the general syntax of inclusion dependency asks each column at lhs to infer from simultaneous column on rhs, for example, for ψ: $\mathrm{R_P}$ [A, B] $\subseteq$ $\mathrm{R'_P}$ [A′, B′], attribute A infer only from A′ not A′ ∪ B′. This property help us consider only *unary* XIND as a normal form of n-ary XIND.

4.1 Data Preprocessing

As mentioned early in section 3.2, XML data representation divides the XML document into a set of related relations $\mathrm{R_P}$ corresponds to an essential tuple class C_p, in our algorithm we need to retrieve the data type used for each repeatable element $(int, string, real)$ to use them later as an attribute data type, this feature is one of the greatest strength of XSD.

The idea is building an association between each data value (v) and attributes [A] that having this value (more precisely constructing a relationship between each leaf node and its direct parent inside the XML tree). The initial phases for relations preparation are:

1. Dividing $\mathrm{R_P}$ attributes $(\mathrm{A} \in \mathrm{R_P})$ for all $\mathrm{R_P}$ relations under XML tree $(\mathrm{R_P} \in \mathrm{T})$ into classes $\mathbb{U}$ shares the same data type t . As known from the basic definition of IND [27] that related

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

attributes between different or even same relation share the same data type.

$$\mathbb{U}_t = \mathrm{R_P}.\mathrm{A} \mid \mathrm{A} \text{ has } t \text{ data type}, \mathrm{R_P} \in \mathrm{T}$$

2. Creating an Active Domain $\mathbb{V}$ for each attribute [A] by selecting a distinct values without redundant from A values.

$$\mathbb{V}_\mathrm{A} = v \in \pi_\mathrm{A}(\mathrm{T}) \mid \mathrm{R_P}.\mathrm{A} \in \mathbb{U}_t, \mathrm{R_P} \in \mathrm{T}$$

3. Constructing a Binary relationship defined as a couple of results of previous two steps, $\mathbb{B} \subseteq \mathbb{V} \times \mathbb{U}$.

$$\mathbb{B} = (v, \mathrm{R_P}.\mathrm{A})$$

To clarify the concept of relations preparation, let's consider XML tree T as a set of $\mathrm{R_P}$ as appeared early in section 2.3.1, and let's consider two relations R_{Order} ,R_{Book}, (see appendix) which contain a sample of data for lack of space. Applying previous steps on these relations will produce next initial results, for example, for *string* datatype, $\mathbb{U}_{str}$: $\{O.Title, B.Title, O.Authar, Genre\ B.Authar, Publisher\}$,$\mathbb{V}_{str}$: {Little Women … , Wiley … , Science … }, finally if the value v = '*First course in Database*' appears in more than one attribute then *B= {(First course in Database, O.Title),(First course in Database, B.Title)}* These result can be summarized as shown next in Table 3.

Table 3: Patterns Tableaus for ψ_3 and ψ_4

$\mathbb{B}_{string}$	
$\mathbb{V}_{string}$	$\mathbb{U}_{string}$
First course…	*O.Title , B.Title,*
Alpha Teach…	*O.Title , B.Title,*
…	…
Wiley	*Publisher*
Ballantine	*Publisher*
Book	*Type*
Article	*Type*
…	…

4.2 Approximate XIND Discovering

Actually, an extraction context can be seen as transactional databases in which attributes are items and values are transactions, moreover inferencing *unary* XIND can be seen as mining association rules whose confidence threshold $\delta = 1$. The major difference between integrity constraints and association rules is that the latter are not schema-level properties and deal with frequency measures instead of real-world probabilities and probabilistic models. Before start arguing about the discovery algorithm, we should introduce the concept of *approximate* inclusion dependencies for XML data model. From the definition of g_3 measure for AFD of relational databases [28] we can hire a modified version g_3' for inclusion dependencies:

$$g_3'\left(\psi: \left((S, [P]) \subseteq (S', [P'])\right), \mathrm{T}\right) = 1 - \frac{\max\{|p|, p \in P(\mathrm{T}')\ s.t.\ \mathrm{T}' \vDash \psi\}}{|P(\mathrm{T})|} \quad 4.1$$

Informally, g_3' is the proportion of value in lhs paths that should be modified (sometimes deleted) from T to produce T′ that satisfied XIND ($\mathrm{T}' \vDash \psi$). g_3' Test asks the user to define the error threshold $\epsilon \in [0,1]$, then approximate XIND satisfies T with respect to ϵ, *iff* $g_3'(\psi, \mathrm{T}) \leq \epsilon$. Now as confidence value indicates to the ratio of correct value where dependency holds in then:

$$Conf(\psi, \mathrm{T}) = 1 - g_3'(\psi, \mathrm{T})$$

As $g_3'(\psi, \mathrm{T}) \leq \epsilon$ Then $T \vDash \psi$ if

$$Conf(\psi, \mathrm{T}) \geq \delta \quad 4.2$$

Formula (4.2) shows the minimal ratio needed to consider a driven dependency satisfies tree under XIND definition, if the user provides a specific thresholds δ, then T satisfy XIND ($\mathrm{T} \vDash \psi$) if $Conf(\psi, \mathrm{T}) \geq \delta$, as a result ($\mathrm{T} \vDash_\delta \psi$). Inference an interesting approximate unary XIND can done just by one pass thought the extraction context as algorithm an in Fig.4 shows.

Approximate XIND Mining $((\mathbb{B}, \mathbb{V}, \mathbb{U}), \delta)$
{
1. $n = 0$; // counter for threshold value
2. $int\ array[\,][\,]$
3. $\mathfrak{U} = \mathbb{U}_t \mid t \in \{int, string, date, \dots\}$ // set of all attributes for all $\mathrm{R_P}$
4. **for each** $\mathrm{A} \in \mathbb{U}_t$ **do**
5. $rhs(\mathrm{A}) = \{< \mathrm{B}, n > \mid \mathrm{B} \in \mathbb{U}_t\}$;
6. **for each** $v \in \mathbb{V}_t$
7. **for each** A s.t. $(\mathrm{v}, \mathrm{A}) \in \mathbb{B}_t$
8. **for each** $< \mathrm{B}, n > \in rhs(\mathrm{A}) \mid ((\mathrm{v}, \mathrm{B}) \in \mathbb{B}_t)$
9. $n_{AB} = n_{AB} + 1$;
10. **for each** $\mathrm{A} \in \mathbb{U}$ **do**
11. **for each** $< \mathrm{B}, n > \in rhs(\mathrm{A}) \setminus (\mathrm{v}, \mathrm{A}) \in \mathbb{B}_t$
12. **if** $((n_{AB}/n_{AA}) \geq \delta)$
13. **then** $\mathrm{XIND} = \mathrm{XIND} \cup \{\mathrm{A} \subseteq \mathrm{B}\}$
14. $\mathfrak{U} = \mathfrak{U} \setminus \{\mathrm{A}, \mathrm{B}\}$
15. **return** XIND , $\mathfrak{U}$

}

Figure 4: XIND Inferencing Algorithm.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

n_{AB} : Support of the dependency.

n_{AA} : Cover of the dependency.

$$Conf(\varphi, \mathrm{T}) = \frac{supp(\varphi, \mathrm{T})}{cover(\varphi, \mathrm{T})} = \frac{n_{AB}}{n_{AA}}.$$

Once the extraction context phase groups relation attributes based on their domain type, *approximate* XIND mining algorithm invoked for each data type, for instance in our case there are three main data types *(int, date, string*). Moreover, the algorithm starts by assigning all attributes to share the same domain type as *rhs* for each attribute (line 4,5), we omit the selector paths and just use the attribute name (field path) to clarifying the context i.e. $\{Genre\}$ is for $\{Book/Genre\}$.

Lines 4, 5 fill rhs for all attributes with each attribute shares same data type, for string data type:
$rhs(B.Title) = rhs(O.Title) = rhs(B.Authar)$
$rhs(O.Author) = rhs(Genre) = rhs(Publisher)$
$= \{< O.Title, 0 >< B.Title, 0 >< O.Authar, 0 >$
$< B.Author, 0 >< Genre, 0 >< Publisher, 0 >\}$.
$\{Type\}$ Column not used here as it has finite domain $\{Book, Article\}$. Then each value v from the extraction context used to update n value, for instance the value $v ='$ First course in Database$'$, appears in two attributes *(*$O.Title$ and $B.Title$*),* so we increased n by one for both of them, whereas not participated attributes remains unchanged. The final results after calculating the number of occurrence for each element value of $string$ data type and between related attribute of R_{Order}, R_{Book} relations can be depicted as a two dimensional array $[lhs][rhs]$, as illustrated next in Table 4.

To see how XIND generated, let's consider the library has 500 books (with 320 titles and 255 authors), and 200 orders requested by students, 80 of them books (with 70 titles and 65 authors) and the remaining are articles. Remember that choosing active domain will return values without redundant as may be many copies of a book available and may be the same book ordered by a number of students.

Table 4: Final Result after XIND Procedure Invoked.

lhs \ rhs	B.T	O.T	B.A	O.A	Gen.	Pub.
B.Title	320	70	0	0	0	0
O.Title	70	200	0	0	0	0
B.Author	0	0	255	65	0	0
O.Author	0	0	65	200	0	0
Genre	0	0	0	0	13	0
Publisher	0	0	0	0	0	176

To formulate XIND we need to assign confidence threshold δ, as large as the confidence value is as less as XIND numbers are; suppose $\delta = 0.75$, this value means at least 75% of the values in dependent relation should appear at referenced relation in order to consider discovered dependency satisfied.

Now as we mentioned the total number of orders are not only books but also article, so no XIND will be discovered using this confidence value as φ: order[Title] ⊆ book[Title] has (70/200 = 0.35) value, but the truth is all book orders are in the book relation and this highlight the important of proposed XCIND, as not all orders should take under consideration to appear in books relations but only whom had Book Type.

Reducing the confidence value ($\delta = 0.25$) will help us producing XIND and false dependencies will be unveiled in next steps.

$$\left(\frac{n_{AB}}{n_{AA}}\right) = \frac{70}{200} \geq \delta$$

Moreover, approximate XIND algorithm will discover three more inclusion dependency between book and order table using the same threshold:

φ: order [Author] ⊆ book[Author].
φ: book[Title] ⊆ order[Title].
φ: book [Author] ⊆ order[Author].

The mining algorithm reduces the search space by eliminating any attribute committed as a participant in any inclusion dependency from conditional attributes, as any attribute cannot participate in both path types (line 14).

4.3 XCIND Main Mining Algorithm

Patterns for inclusion dependencies differ slightly from those assigned for XCFD; as they have two kinds of conditions; *selecting* and *demanding* conditions. Selecting condition appear on lhs relation ($\mathrm{R_P}$) and used to ensure the validity of included dependency whereas demanding condition used on rhs relation ($\mathrm{R'_P}$) and needed to ensure the completeness of the relation.

In this paper, our mining XCIND algorithm focus on selecting condition as their requirement subsume the demanding condition requirement. To change discovered dependencies to conditional ones, we need to search for patterns though non-inclusion attributes; ($\mathrm{R_P}.\mathrm{Attr} \cap \mathfrak{U}$). Mining a set of XCIND pass thought a set of procedures invoked from the main algorithm appear in Fig.5. The output of each procedure used as an input for another procedure until the requested set of rules produced.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Algorithm Mining XCIND ()
Input: XML document as a set of R_p with attributes $a_1 \dots a_n$, supp θ, Conf δ
Output: Set of Minimal XCIND
Preprocessing Phases
1. EX = Preprocessing(set of R_p) // extraction context for each data type in R_p
2. XIND = Approximate XIND Mining (EX, δ) //discover XIND with size =1

Initialize variables steps
3. XCIND = ∅ // stored discovered XCIND
4. Level $l = 1$ // Single attribute set size at lattice
5. $C_l = R_p.\text{Attr} \cap \mathfrak{U}$ // Fill level l_1 candidates with selected attributes of R_1

Iteration steps
{
6. **for each** $\psi : (R_P\,[A]) \subseteq (R'_P, [B]) \in$ XIND
7. XCINDs = XCINDs ∪ XCINDGenerator $(C_l, \psi, \hat{\theta}, \hat{\delta})$
8. MinimalCover (XCINDs)
9. **return** (XCINDs)
}

Figure 5: XCIND Mining Main Algorithm.

The algorithm starts by defining a set of parameters and variables to store values during the algorithm debugging. At level $l = 1$, the algorithm looks only for XCIND which has a single conditional attribute on the *lhs* relation. XCINDGenerator Procedure invoked to form a dependency that has predefined support and confidence thresholds differ from used for XIND. ApriorіGeneration procedure [21] called to generate candidates with a large size.

For a set of discovered approximate XIND, each one will convert to an XCIND using procedure appear in Fig.6. The general idea is threefold: First: computing a JOIN between participant relations (R_P, R'_P) as line 4 depicts, these relations known as dependent relation (R_P) and referenced relation (R'_P). Second: using R'_P attribute to indicate included tuples from not included (line 5). And finally grouping the result using preferred attributes (line 6) which used as a condition attributes (line 3).

After that, each procedure fetched to do a special task, the result back again to the main algorithm. The complexity time for this algorithm is the summation of all sub procedure times.

XCINDGenerator $(C_l, \psi, \hat{\theta}, \hat{\delta})$
{
1. **while** $|l| <> n$ **do**
2. **For each** X $\in C_l$
3. **SELECT** X, $count$(B)
4. **FROM** R_P , R'_P
5. **WHERE** A = B
6. **GROUP BY** X
7. **HAVING** $count(B) \geq Num$ // $(\hat{\theta} * N)$
8. **And** $(cast(count(B))/cast\ (count(A))) \geq \hat{\delta}$
9. XCIND = XCIND ∪ (R_P [A; X]) ⊆ (R'_P [Y])
10. $l = l + 1$;
11. C_l = ApriorіGeneration (C_{l-1})
12. **Return** XCIND;
}

Figure 6: XCIND Generator Procedure.

During the search for patterns, normal query statement will count each tree tuple as a participant, for instance, it will count 200 orders, this case produces useful patterns with completeness option and elapsed less running time. Furthermore, inserting *distinct* option will remove the redundant element and count single element to ensure patterns availability, the called cover option and produce more tangible patterns but consumes more running time. As our main target is discovering patterns for supported XIND, once our algorithm mined an XIND, it moves directly to looking for a pattern for it. More precisely, the final result will show only XIND that has related patterns, other false XIND will be eliminated as final results.

4.3.1 Merging Pattern of XCIND

The final step required ensure the minimality of discovered XCIND is to regroup these pattern so dependencies share same conditions merged together in order to minimize searching processes. These operations are performed by the *MinimalCover* procedure presented in Fig.7.

MinimalCover (Σ)
{
1. **for each** XCIND : $\psi \in \Sigma$
2. $tempLHS = \psi\ (LHS\ elements \cup values)$
3. $tempRHS = \psi\ (RHS\ elements \cup values)$
4. **if** ($temp_{tp}[tempLHS, tempRHS]) \notin \Sigma_{mc}$
5. $\Sigma_{mc} = \Sigma_{mc} \cup temp_{tp}$
6. **else**
7. $match\ (temp_{tp}, \Sigma_{mc})$
8. return Σ_{mc}
}

Figure 7: Minimal Cover Procedure.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

The method *match* checks if the *lhs* value (respectively *rhs*) are used before with the same attribute list. If so, the method merges both dependencies with the *lhs* value (resp. of *rhs*) at the proper place (attributes stored in *lhs* and *rhs*).

5. XCIND DETECTING INCONSISTENCIES

The next step toward verifying proposed XCIND is detecting or identifying responsible data values which cause erroneous and inconsistent, more precisely, detecting XCIND violations. Given a data instance as an XML document and a set Σ of conditional inclusion dependencies on T, finding all paths in T that violate some XCIND in Σ.

In order to test the consistency of an XML database state relative to a given constraint, the system often computes its denial query, extracting those tuples that do not satisfy the constraint predicate in negated form (called the denial form of the constraint); consistency holds if the denial query returns an empty result. The tuples extracted by the denial query represent constraint violations [29].

Detecting Inconsistencies (Σ, T)
{
Input: XML document as a set of R_p,
Set of XCIND, ψ
Output: a set of inconsistencies
$inco$ [] $= \emptyset$ // for discovered inconsistencies
1. **For each** ψ: ($lhs \subseteq rhs$; tp) $\in \Sigma$
2. **For each** (R_p | ($Attr(R_p) \cap$
3. ψ (lhs) $\neq \emptyset$))
4. **For each** (R'_p| ($Attr(R'_p) \cap$
5. ψ (rhs) $\neq \emptyset$))
6. ϵ = SELECT *lhs* FROM R_p , tp WHERE *lhs* NOT IN
7. (SELECT *lhs* FROM R_p , R'_p , tp
8. WHERE $R_p.P = R'_p.P$
9. AND $R_p . lhs = t_p . lhs$)
10. And $R_p . lhs = t_p . lhs$
11. $inco = inco \cup \epsilon$
12. **return** *inco*
}

Figure 8: Detecting Inconsistencies Algorithm.

Inclusion Dependencies (XIND, XCIND) provide different criteria to classify a path as inconsistent from XCFD, any path appears in the dependent paths (lhs) should appear also in the referenced paths (rhs). For example, in the order subtree (relation), there exist a book discovered using generated query based on XCIND ψ: $order\ [Title, Author; Type] \subseteq book\ [Title, Author], tp = 'book'$) Contains following information (Title = 'ipv6 essentials', and Author = 'Silvia Hagen'), unfortunately, this book is not available at book subtree.

The truth is modifying referenced relation (book) asked for inserting new path into the *rhs* relation in order to make dependency satisfied. On the other side, modifying dependent paths relation (order) needs even to delete or update paths values in order to make them obey related dependency. Modifying inconsistencies is out of scope this paper and will be left for future work.

Procedure in Fig.8 checked each XCIND against related relation R_p, R'_p, is there exist a tree tuple appears in lhs of the dependency and not appear at rhs, then it needs an insertion as a requirement of the solution.

6. EVALUATION

6.1 Experimental Conditions

We evaluate our methods with two datasets: synthetic dataset (University.XML) and real dataset (Mondial.XML), summary for both datasets will be shown next in Table 5.

All our experiments were performed on an Intel Core i5-2410M with a CPU clock rate of 2.3 GHz, 8 GB of main memory and running Windows 10 professional. Algorithms were implemented using the C# language with LINQ programming model. SQL Server 2016 was used to perform tests while RDBMS accesses SqlClient.

In order to import needed XML file to the application, we use XML read techniques to load XML file, create the related schema and then shredding XML tree based on pivot paths techniques and finally store relation in the new database. Moreover, XML documents are required to be well-formed to allow application starts working.

Table 5: Dataset Summary.

	Size KB	# of nodes	# of rep paths
University	128	9143	3
Mondial	1,198	49,422	28

6.2 XCIND Mining Algorithm Analysis

Mining good quality rules will effect on the XML data quality model, for this reason, we test our mining algorithms with different *confidence* values;

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

as changing *support* values will affect only the number of paths that dependency based on (degree of validity). The number of discovered dependencies will change consequently as well as running time by changing confidence thresholds as appear in Fig.9 and Fig.10.

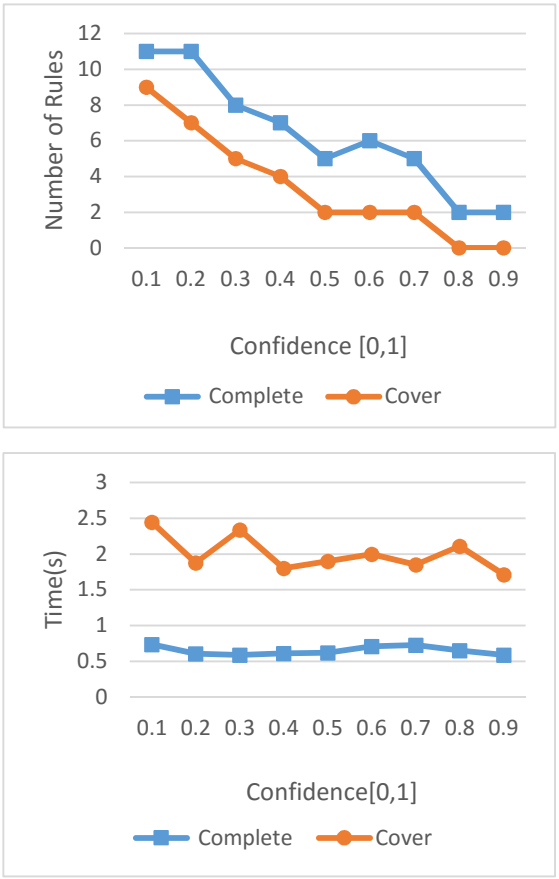

Figure 10: Scalability and Number of discovered XCIND (University data set)

As appear from Fig.9, the number of discovered XCIND decreases by increasing the tableau confidence threshold. Whereas the time still liner, which proves our algorithm of mining conditional inclusion rules regardless the complexity of XML tree (same for Fig.10).

Another factor effects number of dependencies is using the DISTINCT keyword in a HAVING or even SELECT statements to remove duplicates returned by a query for XCIND Generator procedure will enforce the algorithm to choose single value for multiples redundant values, more precisely, instead of having same text node at *lhs, rhs* relations, only one value can ensure the consistency of the dependency on XML tree, Nevertheless, the complexity time increased; due to the sort operation that the SQL enforced to perform.

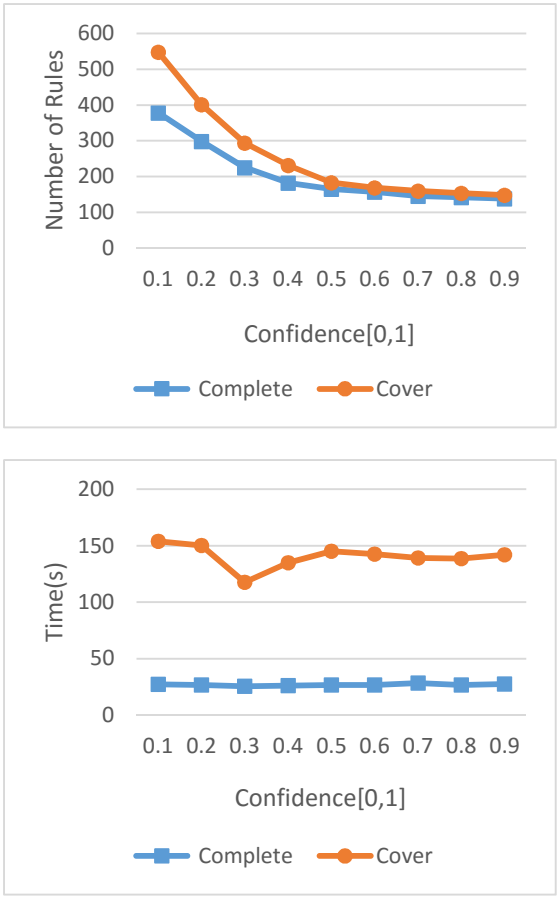

Figure 9: Scalability and Number of discovered XCIND (Mondial data set)

7. CONCLUSIONS

Conditional dependencies played an important role in improving relational databases quality, as XML data model becomes standard for data transferring and integration, XML data quality become more crucial. In this paper, we introduce a conditional version of standard XML inclusion dependencies special for data quality issues. Moreover, a novel mining algorithms to inferring minimal, non-trivial rules are introduced and tested with different confidence threshold. Finally, discovering inconsistencies using mined rules are presented to prove the ability of XCIND to detect inconsistent tree tuples between multilevel inside XML tree.

In conclusion, discovered dependencies will be implemented to discover data inconsistencies from XML tree and will recover inconsistencies as needed.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Proposed discovering algorithms for XCIND and XCFD as future work produced a set of useful dependencies to be used as business rules.

Many major difficulties faced us during this research, for example, not all XML documents are associated with their schema files, moreover, not all XML document contains valuable information, therefore, mined rules looks useless.

This research is the first phase to build a model for improving XML data quality using a set of conditional integrity constraints and to open a new research doors for using these promising rules for others non-structured database like JSON, furthermore, Using enhanced XCIND in more than data cleaning problems, like data publishing and data integration will be left as a future work for this research.

REFERENCES

[1] L. Li, "Data quality and data cleaning in database applications," no. September 2012.

[2] Larisa Bedgood, "How Much is Dirty Data Costing You? " 2015. [Online]. Available: http://www.datamentors.com/blog/how-much-dirty-data-costing-you. [Accessed: 16-Jan-2016].

[3] S. Grijzenhout and M. Marx, "The quality of the XML Web," *J. Web Semant.*, vol. 19, pp. 59–68, 2013.

[4] Computing Research Association, "'Challenges and opportunities with big data,'" 2012.

[5] J. Chen *et al.*, "Big data challenge: A data management perspective," *Front. Comput. Sci.*, vol. 7, no. 2, pp. 157–164, 2013.

[6] Z. Tan and L. Zhang, "Repairing XML functional dependency violations," *Inf. Sci. (NY).*, vol. 181, no. 23, pp. 5304–5320, 2011.

[7] W. Fan, "Data Quality : From Theory to Practice," *SIGMOD Rec.*, vol. 44, no. 3, 2015.

[8] R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems*, 7th Editio. Pearson Education, 2016.

[9] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis, "Conditional functional dependencies for capturing data inconsistencies," *ACM Trans. Database Syst.*, vol. 33, no. 2, pp. 1–48, 2008.

[10] S. Ma, W. Fan, and L. Bravo, "Extending inclusion dependencies with conditions," *Theor. Comput. Sci.*, vol. 515, no. January, pp. 64–95, 2014.

[11] L. T. H. Vo, J. Cao, and W. Rahayu, "Discovering conditional functional dependencies in XML data," in *Proceedings of the Twenty-Second Australasian Database Conference-Volume 115*, 2011, vol. 115, no. 5, pp. 143–152.

[12] M. Hakawati, P. Saad, N. Sabri, Y. YACOB, R. B. AHMAD, and M. S. Salim, "XML INTEGRITY CONSTRAINTS, WHAT'S NEXT?," *J. Theor. Appl. Inf. Technol.*, vol. 92, no. 2, p. 365, 2016.

[13] P. Bohannon, W. Fan, M. Flaster, and R. Rastogi, "A cost-based model and effective heuristic for repairing constraints by value modification," *Proc. 2005 ACM SIGMOD Int. Conf. Manag. data - SIGMOD '05*, p. 143, 2005.

[14] W. Fan, F. Geerts, and X. Jia, "A revival of integrity constraints for data cleaning," *Proc. VLDB Endow.*, vol. 1, no. 2, pp. 1522--1523, 2008.

[15] X. Chu *et al.*, "KATARA: A Data Cleaning System Powered by Knowledge Bases and Crowdsourcing," *Proc. ACM SIGMOD Int. Conf. Manag. Data*, pp. 1247–1261, 2015.

[16] J. Liu, J. Li, C. Liu, and Y. Chen, "Discover dependencies from data - A review," *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 2, pp. 251–264, Feb. 2012.

[17] H. Chen and H. Liao, "Inclusion dependencies for XML," in *2010 International Conference on Information, Networking and Automation (ICINA)*, 2010, pp. 82–86.

[18] M. M. M. Karlinger, M. M. M. Vincent, and M. M. Schrefl, "Inclusion dependencies in XML: Extending relational semantics," *Lect. Notes Comput. Sci. (including Subsea. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 5690 LNCS, no. 9, pp. 23–37, 2009.

[19] M. W. Vincent, J. Liu, and M. Mohania, "On the equivalence between FDs in XML and FDs in relations," *Acta Inform.*, vol. 44, no. 3–4, pp. 207–247, 2007.

[20] W. Fan and F. Geerts, "Foundations of Data Quality Management," *Synth. Lect. Data Manag.*, vol. 4, no. 5, pp. 1–217, 2012.

[21] M. M. Aqel, N. F. Shilbay, and M. Hakawati, "CFD-Mine: An Efficient Algorithm For Discovering Functional and Conditional Functional Dependencies," *Trends Appl. Sci. Res.*, vol. 7, no. 4, pp. 285–302, Apr. 2012.

[22] C. Yu and H. V. Jagadish, "XML Schema refinement through redundancy detection and normalization," *VLDB J.*, vol. 17, no. 2, pp. 203–223, 2008.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[23] Y. Huhtala *et al.*, "Tane: An Efficient Algorithm for Discovering Functional and Approximate Dependencies," *Comput. J.*, vol. 42, no. 2, pp. 100–111, 1999.

[24] S. Fast, I. Mlynkova, and M. Necasky, "On Mining XML integrity constraints," in *2011 Sixth International Conference on Digital Information Management*, 2011, pp. 23–29.

[25] M. Arenas and L. Libkin, "A normal form for XML documents," *ACM Trans. Database Syst.*, vol. 29, no. 1, pp. 195–232, 2004.

[26] J. Vidaković, I. Luković, and S. Kordić, "Specification and Implementation of the Inverse Referential Integrity Constraint in XML Databases," in *Proceedings of the 7th Balkan Conference on Informatics Conference*, 2015, p. 18.

[27] M. A. Casanova, R. Fagin, and C. H. Papadimitriou, "Inclusion dependencies and their interaction with functional dependencies," *J. Comput. Syst. Sci.*, vol. 28, no. 1, pp. 29–59, 1984.

[28] J. Kivinen and H. Mannila, "Approximate inference of functional dependencies from relations," *Theor. Comput. Sci.*, vol. 149, no. 1, pp. 129–149, 1995.

[29] S. Ceri, F. D. I. Giunta, P. L. Lanzi, and P. Milano, "Mining Constraint Violations," vol. 32, no. 1, 2007.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Appendixes

Sample R_{Book} dataset.

R Book					
ISBN	*Title*	*Author*	*Year*	*Publisher*	*Genre*
1592	Alpha Teach Yourself …	Trudy Suggs	2003	Alpha	Science
0671	First course in Database	Jenifer Widom	2007	Wiley	Science
0451	Little Women	Louisa May Alcott	1988	Signet Classics	Science
0345	Protect and Defend	Richard North	2001	Ballantine	Romance
…	…	…	…	…	…

Sample R_{Order} dataset.

R Order			
ID	*Title*	*Author*	*Type*
33	First course in Database	Jenifer Widom	Book
30	Adaptive receiver for data transmission over time-dispersive channels.	Shahid U. H. Qureshi, E. E. Newhall	Article
32	The estimated cost of a search tree in binary words.	Alexey Fedotov, Boris Ryabko	Article
31	Alpha Teach Yourself American Sign Language in 24 Hours	Trudy Suggs	Book
34	ipv6 essentials	Silvia Hagen	Book
…	…	…	…