Discovering XML Conditional Dependencies for Data Quality Issues

Mohammed Hakawati, Yasmin Yacob, Amiza Amir, Jabiry M.Mohammed, and Khalid Jamal Jadaa

Abstract—Extensible Markup Language (XML) is emerging as the primary standard for representing and exchanging data, with more than 60% of the total; XML considered the most dominant document type over the web; nevertheless, their quality is not as expected. XML integrity constraints - especially XFD- plays an important role in keeping the XML dataset as consistent as possible, but their ability to solve data quality issues is still intangible. The main reason is that old-fashioned data dependencies were basically introduced to maintain the consistency of the schema rather than that of the data. The purpose of this study is to introduce a method for discovering pattern tableaux for XML conditional dependencies to be used for enhancing XML document consistency as a part of data quality improvement phases. The notations of the conditional dependencies as new rules are designed mainly for improving data instance and extended traditional XML dependencies by enforcing pattern tableaux of semantically related constants. Subsequent to this, a set of minimal approximate conditional dependencies (XCFD, XCIND) is discovered and learned from the XML tree using a set of mining algorithms. The discovered patterns can be used as a Master data in order to detect inconsistencies that don't respect the majority of the dataset.

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

I. INTRODUCTION

Today, data become the lifeblood of businesses, as different database applications, such as Decision Support Systems, Customer Relationship Management, Data Warehouses, Web Services, and eLearning Systems are being used; beneficial information and knowledge can be gained from considerable amounts of data. However, investigations demonstrate that heaps of such applications fail to run successfully and efficiently, due to many issues, such as poor system design or weak query performance, yet nothing is sure to cause application failure than carelessness of data quality issues [1].

According to reports presented by V12-Data in 2015, the expenses of bad data might be considerably higher than 12% lost revenue. About 28% of individuals who had issues related to the delivery of emails said that customer service has endured accordingly, while 21% experienced reputation damage. The vast majority of the organizations (86%)

admitted that their data might be inaccurate somehow. About 44% of businesses and organizations reported that missing or imperfect data is the most frequent problem alongside obsolete information [2].

The impact of poor data quality can be of three types: Operational (causing customer and employee dissatisfaction and increased costs), Tactical (affecting decision making and causing mistrust), and 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 issues [3], [4].

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.

Grijzenhout & Marx, 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 this document [5].

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 [6], and to start searching for data cleaning approaches for XML (Weis, Monod, & Cedex, 2007) especially approaches based on integrity constraints (IC) [7]–[9].

A. Motivational Example

Consider an XML dataset contains information about university libraries with a finite number of books and articles to be used for the online borrowing system and transferring transactions during daily orders as appearing next in Fig 1. The system will use information stored in the XML file to check available resources (books, articles) and return order's query results.

Book, Order, and Article nodes are three complex type elements with a sequence of elements and data values below them. For instance, the complex element 'Book' represents a set of books; each one has Title, Author, Year, Publisher and Genre as non-key elements and ISBN as a key, same idea shared with Article and Order complex elements. A set of two XML Inclusion Dependencies (XIND) held in the XML tree T and used for schema matching, in addition to a single

Published on January 23, 2020.

Mohammed Hakawati is with CETIC Research Centre, Belgium.
(mohammed.hakawati@cetic.be, mshakawati@hotmail.com)

Yasmin Yacob, Amiza Amir, and Khalid Jamal Jadaa are with the School of Computer and Communication Engineering, University Malaysia Perlis, Campus Pauh Putra, Perlis, Malaysia.

Jabiry M.Mohammed is with the Management & Science University, (MSU), Shah Alam, Malaysia.

XML Functional Dependency (XFD) are defined as follows:

$$XIND_1: (\text{university/orders/order [Title, Author]}) \subseteq (\text{university/library/book [Title, Author]})$$

$$XIND_2: (\text{university/orders/order [Title, Author]}) \subseteq (\text{university/library/journal/article [Title, Author]})$$

$$XFD: (\text{university/library/book [Title]}) \rightarrow (\text{university/library/book [Author]})$$

Fig 1: Sample Library Dataset within XML Document.

The first XIND insists that if an order (text node) is requested by a student, it should find a matching data value (text node) at library books, entirely as the second XIND demands that the request finds a matching in available articles. Given XFD appeals that if two books agree on their Title value, they should share the same Author value. These dependencies are satisfied by a given XML document ($\Sigma \models T$) and require *full* agreement between both sides of the XIND dependency. However, these two inclusion dependencies do not make sense; how can the same order node match nodes from two different XML subtrees at the same time. To remedy this contradiction and without loss of generality, let us add a *conditional* element to *lhs* of the dependencies to become:

$$XIND_3: (\text{university/orders/order [Title, Author; Type = book]}) \subseteq (\text{university/library/book [Title, Author]})$$

$$XIND_4: (\text{university/orders/order [Title, Author; Type = article]}) \subseteq (\text{university/library/journal/article [Title, Author]})$$

XIND₃ asserts that for each order in the orders subtree, if the type element is a book, then there must exist a text node

in the book subtree such that order [Title, Author] = book [Title, Author]. Furthermore, XIND₄ asserts that for each order in the orders subtree, if its *type* element matches 'article', then there must exist a text node in the articles' subtree such that order [Title, Author] = article [Title, Author]. However, this type of dependencies makes a *weaker* assertion and highly applicable to real-world cleaning capabilities over traditional XIND ones [10].

These enhanced dependencies are XIND hold only on a subset of the XML document, which satisfies the *condition* rather than the entire document, also, to be checked only using the related dependency, not both [11]. Another case for XFD, as known, XFD can check the consistency by at least two paths, if there exists a single path contains a wrong value, then, how can an inconsistency in a text node be captured? For instance, when the Title of the book is "First Course in Databases", Then the author will be "Jenifer Widom" not "Thomas M Connolly" as appear in Fig 1. Traditional XFD is not able to discover this violation, adding a pattern tableau will solve the problem [12]. This type is called *constant* XCFD. In this research, a set of approximate constant patterns will be used in order to detect inconsistencies with low frequencies.

$$XCFD: ((\text{university/library/book [Title = "First course in Databases"]}) \rightarrow (\text{university/library/book [Author = "Jenifer Widom"]}))$$

II. LITERATURE REVIEW

Obviously, in recent years, XML IC's have attracted the attention of scientist who aims at ensuring data consistency as it is vital and indispensable [13]. The expression "Integrity Constraint" in XML is regularly used to mean extensions of relational ICs, such as Primary Keys, Foreign Keys, Functional Dependencies, and Inclusion Dependencies, which depends principally on the equality of data values within the same subtree or between different subtrees. Traditional XML Dependencies (XFD, XIND, XAFD) have limit capability toward cleaning the XML tree effectively. However, recently, a conditional version of these dependencies presented to overcome their limitation and to open a new era of cleaning approaches for XML cleaning [11], [12]. Sections below cover the basics of these conditional dependencies.

A. XML Conditional Dependencies

Inspiration from the relational database; conditional dependencies (CFD, CIND) were presented to overcome relational traditional dependencies (FD, IND) limitations, especially data quality issues [14], [15]. The conditional dependencies own more quality characteristics that make them directed toward data cleaning such as covering a subset of the dataset. Furthermore, these dependencies proved their efficiency in eliminating inconsistencies from relational databases and detecting more inaccurate tuples and fields within tuples over traditional dependencies. Also, cleaning approaches that adopted these dependencies are considered the most used techniques in the last ten years,

besides crowdsourcing and knowledge base cleaning approaches [16].

Adding some conditional values as *pattern tableaux* to traditional dependencies will make it acquire more required characteristics. However, it will not make things easier in terms of the complexity of time and validity, also, the number of CD that is equal to a single traditional one can also increase. Pattern tableau can be defined as a subset of tree tuples in which the underlying conditional integrity constraints hold. It concisely summarizes subsets of the XML values that mostly satisfy or fail the constraint.

Definition 1: Let $\varphi : (S, [P_1 \dots P_n \rightarrow P_{n+1}])$ as a standard XFD, a pattern tableau T_p of φ restricts *lhs* fields ($P_1 \dots P_n$) over the conditional elements $P_c[L]$ and *rhs* field (P_{n+1}) over the conditional element $P_c[R]$. For each path $[A]$ in $P_c[L]$ or $P_c[R]$, and each single $t_p \in T_p$; $t_p[A]$ is either a constant 'a' from $dom(A)$ or an arbitrary value '-'. XML tree tuple [17], $(t \in T)$ matches $(t_p \in T_p)$, written as $(t \approx t_p)$; $\forall P_c[L], P_c[R] \in t$ and, $\forall t_p[L], t_p[R] \in T_p$; $P_c[L] = t_p[L]$ and $P_c[R] = t_p[R]$.

The pattern tableau is divided into *lhs* ($t_p[L]$) with all $P_c[L]$ paths and *rhs* ($t_p[R]$) with all $P_c[R]$ paths as depicted in Table 1. Both sides of the tableau can be left empty specifying no restriction on any element. In this research, $P_c[L], P_c[R]$ paths over *lhs*, *rhs* subtrees will be called *Conditional Paths*, and separated by a semicolon (;) from field paths, below are the pattern tabulate for aforementioned example:

XIND₃: (university/ orders/
order [Title, Author ; Type]) $\subseteq$ (university/library/
book, [Title, Author]) , T_1).
XIND₄: (university/ orders/
order [Title, Author ; Type]) $\subseteq$ (university/library/
article, [Title, Author]) , T_2).
XCFD: (university/library/book[nil ; Title]) $\rightarrow$
(university/library/book, [nil ; Author]) , T_3).

TABLE 1: PATTERN TABLEAUS FOR CONDITIONAL DEPENDENCIES.

T ₁	Type	nil	T ₂	Type	nil
	Book	-		Article	-
T ₃	Title		Author		
	First course in Databases		Jenifer Widom		

1) XML Conditional Functional Dependencies

XCFD emerges from relational CFD, traditional dependencies were basically introduced for schema design enhancement, which is a vital part of enhancing schema quality. However, for sure it is not adequate to consider schema quality and disregard data quality. Authors in [12] conduct a conditional copy of XFD called XML Conditional Functional Dependency (XCFD), and change XFD notation to convey new rules.

Definition 2: Using basic definitions mentioned in [17], consider XML data tree as $T = (V, e, lab, val, v_r)$ and XSD as $S = (E, A, T, root)$, XCFD is an expression of the

form: $\varphi = pv : \{e\}, \{X\} \rightarrow \{Y\}$, Where:

- pv : is the Pivot path.
- e : is the conditional part and has a form of $(ex_1 * ex_2 * \dots * ex_n)$, where ex_i is a Boolean expression associated with a conditional element, $*$ is an operator either AND ($\wedge$) or OR ($\vee$).
- $\{X\} \rightarrow \{Y\}$: is a standard XFD form.

The conformation between the tree and XCFD $T \models \varphi$, can be 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 an entire document as mentioned earlier.

2) XML Conditional Inclusion Dependencies

XML Conditional Inclusion Dependencies (XCIND) syntax [11] is too close to strong path notations [18], as this notation is the closest notation to the relational notation and the most explicit description of XML Inclusion Dependencies.

Definition 3: XCIND ψ is a pair:

$((S, [P ; P_c]) \subseteq (S', [P' ; P'_c]), T_p)$, where:

- $((S, [P]) \subseteq (S', [P']))$: 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 non-empty 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 that end with an attribute or text nodes.
- $[P_c]$ and $[P'_c]$: are two non-empty sequences of paths that end with text match values of pattern tableaux, and it is called Conditional Paths (elements).
- T_p : is a pattern tableau containing 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 $Dom(A)$.

Referring to the motivational example earlier, XCIND shown below is related to pattern tableau titled T_1 . As seen in Table 2, the pattern tableau, which looks like a contingency table, contains the values that should match to make the dependency hold.

XIND₃: (university/ orders/
order [Title, Author ; Type]) $\subseteq$ (university/library/
book, [Title, Author]) , T_1)

TABLE 1: PATTERN TABLEAU T_1 FOR XIND₃.

	Title	Author	Type	Title	Author
T ₁	-	-	Book	-	-

III. METHODOLOGY

The idea behind tree tuple is to find a relative representation of the XML tree as a relational data model. Some of the important key points about the mapping phase are: firstly, tuples hold related information can be easily accessed by different DBMSs, gives a clearer conceptual view of the XML dataset, and composed values can be compared and verified effectively [19], [20]. secondly, discovered patterns take tableau form, so it becomes much easier to handle XML tree as tables form and combine them at advanced levels.

Mapping XML tree can be done in many ways. Firstly, 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 number of columns (Attributes). Therefore, a direct result of this mapping is the tremendous complexity time [21]. The second way is using a semi-normalized form by shredding XML into many related relations using the concept of tuple class, this representation uses the concept of schema normalization and reduces the complexity of time and space. The aforementioned sample dataset is represented by the following table schemas.

$R_{Library} = [\text{@key, Parent @key, Info/Location, Info/Phone}]$,
 $R_{Journals} = [\text{@key, Parent @key, Name, IF}]$
 $R_{Article} = [\text{@key Parent @key, DOI, Author, Title, Journal, Year}]$,
 $R_{Book} = [\text{@key, Parent @key, ISBN, Title, Author, Year, Publisher, Genre}]$
 $R_{Order} = [\text{@key, Parent @key, OID, Title, Author, Type}]$

A. Discovering Patterns Tableaus for Conditional Dependencies

Since the primary objective of this research is improving the data quality, discovered data quality rules that *obey* dataset schema might not be satisfied in the related instance. More precisely, fully agreed dependencies (XCFD, XCIND) are considered as business rules to prevent inconsistent values from entering the database. On the other side, discovering inconsistencies from the *existing* database requires the definition of some error measure (ϵ) for the satisfaction of patterns; the difference between full and approximate dependencies shows spurious data as a result.

To mine *approximate* dependencies, a *confidence* threshold is required for the identification of the percentage of right values. On the other side, conditional dependencies hold on a subset of the dataset. Therefore, it requires a *support* threshold to identify the percentage of the dataset taken into consideration. Both thresholds will be hired as a set of *approximate conditional* dependencies (XCFD, XCIND) is required as a preceding step for data cleaning. To sum up, a set of *approximate* data dependencies, which need not be true for every instance, will be discovered and inspected to capture data inconsistencies with predefined minimum *support* and *confidence* thresholds values.

1) Interested Pattern Tableaus

The range of discovered patterns from the XML tree T under the new notions and the new algorithms are quite broad. Apparently, it is not a good idea to return the set of all possible rules; some unnecessarily *trivial* and *non-minimal* dependencies can also appear. Therefore, a minimum of canonical cover (non-redundant and minimal set of XCFDs and XCINDs) are taken into consideration, while other dependencies can be driven using implication rules. Moreover, real-life data are often dirty, contain errors and noise. Thus, to exclude patterns that match errors and noise only, a small set of tableau patterns with both high *support* threshold θ (number of paths should match) and high *confidence* value δ (few exceptions) in XML tree are taken into account.

a) Non-Trivial Dependencies

Data dependency is called *trivial* if either its *rhs* as a single path belongs to *lhs* paths i.e. $P_y \in \{P_{x1} \dots P_{xn}\}$ such that XFD $\varphi : \{P_{x1} \dots P_{xn}\} \rightarrow P_y$, or if it does not hold on any subset of T. Mining *non-trivial* dependencies is considered an important aspect for all data models, because it reduces the search space and as a result decreases the complexity time and memory consumption. However, XML dependency notations require additional features to classify the discovered dependency as *non-trivial*. These features do not allow null values to any path that belongs to the *lhs* of XFD. Also, the dependency should rely on an essential tuple class C_p and finally, *rhs* paths always match to a descendant node of the tuple of pivot node n_p .

b) Minimal Dependencies

A dependency (XCFD or XCIND) is considered a left reduced or minimum *lhs*, if the removal of any field path from its *lhs* makes the dependency not satisfied, i.e., for any $G \subseteq (P_{x1} \dots P_{xn}), T \not\models (\{P_{x1} \dots P_{xn}\} / \{G\} \rightarrow P_y)$. This condition assures the *minimality* of dependency.

Additional condition for pattern tableaus is required to enforce the minimality, for any $T_p \preceq \widetilde{T_p}, T \not\models \psi : ((S, [P; P_c]) \subseteq (S', [P'; P'_c]), \widetilde{T_p})$. The ordering symbol ' $\preceq$ ' requires that constant pattern tableau ($t_p(p) = 'a'$) is not *upgradable* to the variable pattern ($t_p(p) = '-'$); this is called the minimality of pattern tableaus.

c) Frequency of the Dependency

Before any argument about the *support* of the dependency begins, the *cover* of the dependency should be defined. The cover can be defined as a set of paths under given tree tuple as *lhs* of the dependency that match the *lhs* of a given pattern tableau divided by the total number of the paths of the tree tuple $|t|$.

$$Cover(\varphi, T) =$$

$$\frac{\{path | path \in tree tuple(t) \wedge path[lhs] \approx t_p[lhs]\}}{\{all path \in tree tuples(t)\}} = \frac{|lhs(\varphi)|}{|t|} \quad 3.1$$

The proportion of paths in a tree tuple that obeys a specific dependency (pattern tableau) is known as the *support* of that dependency. In other words, a dependency can be considered as *valid* if its support is superior to a predefined requested threshold ($Supp(\varphi, T) \geq \theta_{tp}$).

$$Supp(\varphi, T) = \frac{\{path | path \in tree tuple(t) \wedge path[lhs, rhs] \approx t_p[lhs, rhs]\}}{\{all path \in tree tuples(t)\}} = \frac{|\varphi|}{|t|} \quad III.1$$

Therefore, the minimum number of paths required to consider a dependency valid with θ_{tp} as a percentage is:

$$Num(p) = \theta_{tp} * |t| \quad III.2$$

d) Confidence of the Dependency

The *confidence* test is considered one of the highest tests to ensure the *validity* of a given dependency because it requires finding the percentage of the paths in which *lhs* and *rhs* of the dependency appear together and then matching related pattern tableau on the percentage of *lhs* that also matches the pattern tables and appear alone. If *lhs* always occurs with *rhs*, then the confidence value is one, and this looks like traditional XML dependencies.

$$Conf(\varphi, T) = \frac{Supp(\varphi, T)}{Cover(\varphi, T)} \quad III.3$$

XFD has a *support* value of $\theta = 1$, whereas XCFD with a specific pattern tableau has *supported*: $0 \leq \theta_{tp} \leq 1$. Furthermore, an exact XFD or XCFD is required to hold with a *confidence* equal $\delta = 1$, whereas an approximate XFD or XCFD is held with *confidence* within $0 \leq \delta_{tp} \leq 1$. The above properties for pattern tableaux are required together to define a set of the *conical cover* of patterns. In other words, discovered conditional dependencies on T with predefined thresholds θ_{tp}, δ_{tp} , is a set of minimal, *k*-frequent patterns hold on tree T.

B. Discovering XCFD Pattern Tableaus

Once the XML document Tree T is figured out as temporary tables R_p , mining a set of XCFDs involves a set of procedures invoked from the main algorithm 1: Mining XCFD. The pseudo-code of these procedures will be omitted in this paper to the lack of space.

The algorithm starts by defining a set of parameters and variables to store values during the algorithm debugging. After that each temporary relation builds a containment lattice [20], and calculates the partitions for its attributes, however, only non-key attributes participate in the

containment lattice. Looking back at the library example, consider a temporary table R_{Book} with a finite set of columns (*ISBN*, *Title*, *Author*, *Year*, *Publisher*). Fig 2 shows how these columns are connected using *edges* as a potential relationship.

Fig2: Book Search Containment Lattice with $(2^n - 1)$ Nodes using *n* Attributes.

The set of candidates at level $l = 1$ are filled by all single attributes of R_p , and only attributes that contain data (non-key) are checked and added using the *CandSelect* procedure (line 3), while others are used for the XML reconstructing issue. After assigning l_1 candidates, their partitions are calculated using *Singleton Calculate Partition* procedure at line number 4. These candidates are considered as initial XCFD's *lhs* that are connected by an edge with *rhs* candidates at the level l_2 , which will be generated and partitioned at lines 7 and 8 using the procedures *AprioriGeneration* and *CalculatePartition* respectively. Finally, The main algorithm passes partitions (Π_X, Π_Y) for two candidates at two consecutive levels (C_l, C_{l+1}) with a predefined tableau *support* and *confidence* thresholds θ_{tp}, δ_{tp} to the XCFD Generator procedure.

1) XCFD Validity

Consider $\varphi: X \rightarrow Y$ as an XCFD; since the proposed algorithm represents tree tuple values as equivalent classes Π , and the research aim is detecting data violations; then only X partition class x_i , that split into two or more classes y_i in Y are considered to discover a set of *approximate* XCFD.

Algorithm 1: Mining XCFD ( )

Input: XML document as a set of R_p with attributes $a_1 \dots a_n, tb_{supp}(\theta_{tp}), tb_{conf}(\delta_{tp})$

Output: Set of Minimal XCFD (Σ)

For each $R_p \in T$ **do**

Initialize variables steps

1. XCFD { } = $\emptyset$ // Stored discovered XCFD
2. Level $l = 1$ // Single attribute set size at lattice
3. $C_1 = \text{Cand Select}(R_p)$ // Fill level l_1 candidates with selected attributes of R_p
4. $GP_1 = \text{SingletonCalculatePartition}(C_1, T)$ // Calculate Partition for C_1 at $l = 1$

Iteration steps

- ```

{
5.   while $l < n$ do
6. $l = l + 1$;
7. $C_l = \text{AprioriGeneration}(C_{l-1})$ // Generate attribute set in lattice
8. $GP_l = \text{CalculatePartition}(C_l)$ // Generated Partition
9.   XCFDs = XCFDs $\cup$ XCFDGenerator($GP_{l-1}, GP_l, \theta_{tp}, \delta_{tp}$)
10.   Minimal Cover (XCFD)
11.   return (XCFD);
}

```

Initially, the concept of XCFD using a partition, which was adopted by [12], needs to be introduced. For any two-element sets in the containment lattice (X, Y), a class  $x_i$  in  $\Pi_X$  is *subsumed* by  $y_i$  in  $\Pi_Y$ , if  $x_i$  is a *subset* of  $y_i$  or *exactly equal*. Let  $\Omega_X \subseteq \Pi_X$  represent all subsumed classes of  $\Pi_X$ , each  $x_i \in \Omega_X$  presents *full* XCFD; its *lhs* contains a set of paths that are considered *conditional* paths  $C_p$  if the same class  $x_i$  appears in its partitions, otherwise considered a variable field path. Moreover, *rhs* path Y can even be  $C_p$  or field path-based on the existence of a class  $a_i$ , such that  $(a_i \in \Pi_Y) \subseteq \Omega_X$ . Consider two elements candidates (Publisher, Genre),

$\Pi_P = \{ \dots \{116, 117, 118, 119, 120, 121, 122\}, \dots, \{189, 194, 216\} \dots \}$   
 $\Pi_{P,G} = \{ \{116, 121\}, \{117, 118, 119, 120, 122\}, \dots, \{189, 194, 216\} \dots \}$

To see if there a relationship between these elements, their partition should be inspected (not full for lack of space). The subsumed classes -in this case- is  $\Omega_X = \{ \{189, 194, 216\} \}$  with a length equal or superior tableau

support threshold  $\theta_{tp}$ . As the same class is common between  $\Pi_X$  and  $\Pi_Y$ , then there exists a constant XCFD:

$\varphi: [\text{Publisher} = 'Aladdin' \rightarrow \text{Genre} = 'Science']$ .

Consider  $\Lambda_X$  presents non-subsumed classes of  $\Pi_X$ , all classes  $x_i \in \Lambda_X$  and  $x_i \notin \Omega_X$  is not considered as potential full XCFD. Nevertheless, they are still needed to mine approximate XCFD which contains some inconsistencies. Each of these  $x_i$  mapped into two or more classes  $y_i$  in  $\Pi_Y$  indicating that the X value occurs with different Y values. For instance,  $x_{116} = \{116, 117, 118, 119, 120, 121, 122\}$  divided into  $y_{116} = \{116, 121\}$  and  $y_{117} = \{117, 118, 119, 120, 122\}$ .

Let  $y_{max}$  represents the largest  $y_i$  class ( $y_{117} = \{117, 118, 119, 120, 122\}$ ),  $y_{min}$  represents the smallest  $y_i$  class ( $y_{116} = \{116, 121\}$ ) and recall that  $\theta_{tp}$  is the minimum support and  $\delta_{tp}$  is the minimum confidence level. If  $|y_{max}| \geq (\theta_{tp} * n)$  and  $\exists y_{min}$  such that  $|y_{min}| < ((1 - \delta_{tp}) * x_i)$ , and  $|y_{min}| \neq |y_{max}|$ , then  $y_{min}$  paths are reported as a violation for an XCFD  $\varphi: [lhs = x_i \rightarrow rhs = y_{max}]$ . For instance,  $v_{116} = 'Drama'$  as a Genre value is a violation to approximate XCFD  $[\varphi: \text{Publisher} = 'Berkley' \rightarrow \text{Genre} = 'Science']$ .

**Procedure: XCFD Generator ( $GP_{l-1}, GP_l, \theta_{tp}, \delta_{tp}$ )**

- ```

{
1.   list Y = [ ] //classes used for discovering prior rules
2.   for each edge X  $\in C_{l-1}$ , Y  $\in C_l$  ()
3.     if  $|\Pi_X| = |\Pi_Y|$  then XFD: X  $\rightarrow$  A | A  $\in (Y - X)$ , remove X and all its supersets from lattice
4.     else
5.       for each  $x_i \in \Pi_X$  |  $x_i \in \Lambda_X$ 
6.         for each  $y_i \in \Pi_Y$  |  $y_i \subseteq x_i$ 
7.           if  $(|y_i| \geq \theta_{tp})$  and  $(\frac{|y_i|}{|x_i|} \geq \delta_{tp})$ 
8.             then XCFD = XCFD  $\cup$   $\varphi: [x_i = v_{x_i} \rightarrow y_i = \{v_{y_i}\}]$ 
9.             if  $(x_i \in Y)$ 
10.              update  $t_p(v_{x_i}, v_{y_i})$  set  $v_{y_i} = v_{y_i} \cup \widehat{v_{y_i}}$ 
11.            else
12.               $T_p = T_p \cup t_p(v_{x_i}, v_{y_i})$ 
13.               $Y = Y \cup x_i$ 
14.            else
15.              break;
16.   Return XCFD; }

```

Let y_{max} represents the largest y_i class ($y_{117} = \{117,118,119,120,122\}$), y_{min} represents the smallest y_i class ($y_{116} = \{116,121\}$) and recall that θ_{tp} is the minimum support and δ_{tp} is the minimum confidence level. If $|y_{max}| \geq (\theta_{tp} * n)$ and $\exists y_{min}$ such that $|y_{min}| < ((1 - \delta_{tp}) * x_i)$, and $|y_{min}| \neq |y_{max}|$, then y_{min} paths are reported as a violation for an XCFD $\varphi: [lhs = x_i \rightarrow rhs = y_{max}]$. For instance, $v_{116} = 'Drama'$ as a Genre value is a violation to approximate XCFD $[\varphi: Publisher = 'Berkley' \rightarrow Genre = 'Science']$.

XCFD Generator procedure is invoked from the main algorithm to validate elements partitions to see if it has the ability to generate an XCFD rule or not. Unless both sides of the edge share identical partitions, then a non-strict relationship (XFD) needs to be checked (line 3). If there is y_i such that $|y_i| \geq \theta_{tp}$ and there is x_i which has paths more than y_i and less than the *allowed* confidence threshold δ_{tp} , then an *approximate* XCFD is created using line 8.

Let Y be the set $x_i \in \Lambda_X$ used earlier to generate an *approximate* XCFD, if another partition $y_i \in \Pi_Y$ is mapped to the same x_i and has satisfied support value θ_{tp} , then there is no need to create a new dependency, rather they should just be *merged* to produce a richer information rules whose *lhs* value is mapped to more than one *rhs* value (lines 9-14). These two dependencies can be merged into a single dependency and used "AND" operation to separate *rhs* values and to generate a disjunction query statement. This concept reduces the number of produced dependences and will prevent the contradiction of two rules as their *lhs* matches and *rhs* do not.

2) Merging Pattern for XCFD

Procedure XCFD Generator, on the line (10), merge rules that share the same metadata based on the following cases: If two XCFD share same *lhs*, *rhs* selectors, same *lhs* conditional paths with the same patterns, same *rhs* conditional path with different patterns, then these dependencies can produce a dependency with more than pattern tableau:

$$\begin{aligned} \varphi_1: ((\text{book}, [\text{Publisher} = \text{"Little Brown"}]) \rightarrow (\text{book}, [\text{Genre} = \text{"Science fiction"}])) \\ \varphi_2: ((\text{book}, [\text{Publisher} = \text{"Little Brown"}]) \rightarrow (\text{book}, [\text{Genre} = \text{"Science"}])) \end{aligned}$$

For instance, the above dependencies φ_1, φ_2 shared *book* element as *lhs*, *rhs* selectors, *Publisher* as an *lhs* conditional paths, "Little Brown" as a pattern value and shared *Genre* as an *rhs* conditional path, then the following dependency appears after the merging;

$$\varphi: ((\text{book}, [\text{Publisher} = \text{"Little Brown"}]) \rightarrow (\text{book}, [\text{Genre} = \text{Science}, \text{Science fiction}])).$$

The other case, if two XCFD share same *lhs*, *rhs* selectors same *lhs* conditional paths with different patterns, same *rhs* conditional path with the same patterns, then these dependencies can produce a dependency with more than pattern tableau:

$$\varphi_3 = ((\text{book}, [\text{Author} = \text{"Jane Austen"}]) \rightarrow (\text{book}, [\text{Genre} = \text{Science}])).$$

$$\varphi_4 = ((\text{book}, [\text{Author} = \text{"Jack Canfield"}]) \rightarrow (\text{book}, [\text{Genre} = \text{Science}])).$$

Then the following dependency appears after the merging process

$$\varphi = ((\text{book}, [\text{Author} = \text{"Jane Austen"}, \text{"Jack Canfield"}]) \rightarrow (\text{book}, [\text{Genre} = \text{"Science"}])).$$

C. Discovering XCIND Pattern Tableaus

XCIND discovering issue can be expressed as follows: Given an XML tree T over XSD S , find a cover set Σ of all XCIND $\psi: ((S, [P; P_c]) \subseteq (S', [P'; P'_c])), T_P$ such that $T \models \psi$ 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 R_P , and discovers a set of approximate XIND with confidence thresholds δ , and then produces a set of minimal, non-trivial conditional inclusion. The reasons why the discovering of traditional *approximate* XIND needs before the process of searching for XCIND begins are the need to find common elements between subtrees to generate patterns later. Moreover, any attribute of R_P (repeatable path) cannot participate in both types of paths (Conditional and Field). However, in this article, we will omit the Mining XIND phase and considered a set of traditional XIND dependency is already available, refer to [11] to see the full XCIND mining algorithm.

Algorithm 2: Mining XCIND ( )

Input: XML document as a set of R_P with attributes $a_1 \dots a_n, tb_{supp}(\theta_{tp}), tb_{conf}(\delta_{tp})$

Output: Set of Minimal XCIND

Initialize variables steps

1. XCIND [ ] = $\emptyset$ //List to store discovered XCIND
2. Level $l = 1$ // Single attribute set size at lattice
3. $C_l = R_P.\text{Attr} \setminus \mathcal{U}$ // Fill level l_1 candidates with selected attributes of R_1

Iteration steps

- ```
{
4.   for each $\psi : (R_P [A]) \subseteq (R'_P, [B]) \in \text{XIND}$
5.   XCIND = XCIND $\cup$ XCINDGenerator ($C_l, \psi, \theta_{tp}, \delta_{tp}$)
6.   MinimalCover (XCIND)
return (XCIND)
}
```

Patterns of inclusion dependencies differ slightly from those discovered during the XCFD mining algorithm as they have two kinds of conditions; *selecting* and *demanding* conditions. Selecting condition appears on *lhs* subtree ( $S$ ) and is used to ensure the validity of included dependency, whereas demanding condition is used on *rhs* subtree ( $S'$ ) to ensure the completeness of the subtree.

The proposed XCIND mining algorithm focuses on the selecting condition because their requirement subsumes the demanding condition requirement. To convert discovered dependencies (XIND) to a conditional one (XCIND) using the XCIND mining algorithm, the search for patterns though non-inclusion attributes are required. The full algorithms for discovering XCIND are omitted in this paper for the lack of space, however, all details presented in [11].

#### D. Pattern Tableaus Table

A set of *approximate* XCFD and XCIND will be mined using algorithms mentioned earlier on. The question is how all discovered patterns can be arranged in a manner that they can be accessed for the next cleaning phase. As known, each discovered conditional rules would associate with a pattern tableau that contains the context of this dependency. As a result, the number of patterns will be huge (as large as discovered dependencies). To solve this problem, a new concept associated with the storage of patterns called *ConstraintsPatternTableaus* table is introduced in this research. It contains both metadata about dependencies and pattern data as well.

Table 4 illustrates the scheme for the proposed table that holds all information about all discovered XML condition dependencies (XCFD, XCIND). To show how this summary table is created, the requirement of metadata for both dependencies and what pattern information is needed will be explained.

TABLE 4: CONSTRAINTS PATTERN TABLEAUS.

| Constraint Type | lhs Selector | lhs Fields    | lhsCond Paths | lhsCond Values | rhs Selector | rhs Fields    | rhsCond Paths | rhsCond Values |
|-----------------|--------------|---------------|---------------|----------------|--------------|---------------|---------------|----------------|
| XCFD            | Book         |               | Publisher     | 'Berkeley'     |              |               | Genre         | 'Science'      |
| .               |              |               |               |                |              |               |               |                |
| .               |              |               |               |                |              |               |               |                |
| .               |              |               |               |                |              |               |               |                |
| .               |              |               |               |                |              |               |               |                |
| XCIND           | Order        | Title, Author | Type          | 'Book'         | Book         | Title, Author |               |                |
| XCIND           | Order        | Title, Author | Type          | 'Article'      | Article      | Title, Author |               |                |

For each XCFD  $\phi$  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  $\psi$ , *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.

#### IV. IMPLEMENTATION AND DISCUSSION

In order to test the effects of varying tableau support and tableau confidence thresholds on the number of discovered approximate XCFD, one of these thresholds should be fixed and the other one should be varied. The support value for each dependency related to the number of tree tuples in  $R_p$ , for instance, support value  $\theta_{tp} = 0.25$  for a relation with (400) tree tuples will produce only dependencies with (100) supported tree tuples and above. On the other side, varying confidence values will produce a different number of XCFDs based on the number of distinct values. For instance, the algorithm may not discover any dependency with  $\delta_{tp} =$

0.65, whereas two dependencies sharing the same metadata with  $\delta_{tp} = 0.4$  may be discovered, and therefore, after merge operation, both dependencies will produce  $\delta_{tp} = 0.8$  as a dependency confidence (if exist).

Fig 3.A shows the number of XCFD rules mined  $\phi$  when the support threshold  $\theta_{tp}$  varies within the range [0.001 — 0.01], and the tableau confidence fixed at  $\delta_{tp} = 0.4$  with 3 datasets (DBLP, SigmodRecord, Mondial). whereas Fig 3.B shows the same results when the confidence threshold  $\delta_{tp}$  varies within the following range [0.1 — 1], and the tableau support fixed to be  $\theta_{tp} = 0.025$ . As expected, the number of dependencies decreases when the support or confidence value increases; that is because dependencies that have pattern tableaus covered required support thresholds would appear as rules, otherwise, they are considered as *less* important rules.

The total time required for running the mining XCFD algorithm with any support and confidence values is the sum of all subprocedures. Firstly, the algorithm will spend  $((2^n \cdot |t|) \cdot |R_p|)$  for partitioning, and after that, the XCFD Generator will search for rules and perform  $(n/2)$  operations. Therefore, the total bound is  $(|t| \cdot (n/2) \cdot 2^n) \cdot |R_p|$ , and the worst-case complexity is



exponential,  $O(2^N)$ ). The aim of the AppXCFD algorithm is discovering a minimal set of approximate XCFD in order to detect inconsistencies hidden underneath rather than use it as association rules.

Fig 4. A, B shows the number of XCIND discovered by varying tableau support and confidence threshold (different

cases for Complete and Covering) [11]. As expected, the number of produced rules decreases when the tableau thresholds increase. XCIND mining algorithm checks all generated queries for all XINDs and only XIND rule that matched pattern tableaux and has supported equal or large threshold is taken into consideration.

Fig 2: Number of Discovered Patterns for XCFD.

Fig 3: Number of Discovered Patterns for XCIND.

## V. CONCUSSION

The main contribution committed in the research was providing algorithms to discover a minimal set of XCFD and XCIND. Firstly, an algorithm presented to mine and discover a minimal set of constant approximate XCFD between XML elements under the same pivot path. Moreover, a set approximate XCFD is able to detect almost all inconsistencies from the existed dataset unlike full XCFD dependencies, which can prevent inconsistencies during the insert operation.

The scalability shows that the proposed algorithm for mining XCFD or XCIND behaves exponentially concerning the total number of non-repeatable elements with an obvious significance to the number of tree tuples gathered from an XML tree.

## 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 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

- [1] S. Juddoo, "Overview of data quality challenges in the context of Big Data," in *2015 International Conference on Computing, Communication and Security, ICCCS 2015*, 2016, pp. 1–9.
- [2] L. Bedgood, "How Much is Dirty Data Costing You?," 2015. [Online]. Available: <https://www.linkedin.com/pulse/how-much-dirty-data-costing-you-larisa-bedgood/>. [Accessed: 16-Jan-2016].
- [3] N. Laranjeiro, S. N. Soydemir, and J. Bernardino, "A Survey on Data Quality: Classifying Poor Data," in *Proceedings - 2015 IEEE 21st Pacific Rim International Symposium on Dependable Computing, PRDC 2015*, 2016, vol. 10, no. November, pp. 179–188.
- [4] L. Li, "Data quality and data cleaning in database applications," vol. U639248, no. September, p. 1, 2012.
- [5] S. Grijzenhout and M. Marx, "The quality of the XML Web," *J. Web Semant.*, vol. 19, pp. 59–68, 2013.
- [6] 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.
- [7] M. Švirec and I. Mlýnková, "Efficient Detection of XML Integrity Constraints Violation," *Commun. Comput. Inf. Sci.*, vol.

- 293 PART 1, pp. 259–273, 2012.
- [8] H. Hamrouni, Z. Brahmia, and R. Bouaziz, “An Efficient Approach for Detecting and Repairing Data Inconsistencies Resulting from Retroactive Updates in Multi-temporal and Multi-version XML Databases,” in *Advances in Intelligent Systems and Computing*, vol. 312, Cham: Springer, 2015, pp. 135–146.
- [9] Z. Tan and L. Zhang, “Improving XML Data Quality with Functional Dependencies,” in *International Conference on Database Systems for Advanced Applications*, 2011, no. 60603043, pp. 450–465.
- [10] Z. Tan, L. Zhang, W. Wang, and B. Shi, “XML data exchange with target constraints,” *Inf. Process. Manag.*, vol. 49, no. 2, pp. 465–483, 2013.
- [11] M. Hakawati, Y. Yacob, R. A. A. Raof, A. Amir, J. M. Mohammed, and E. S. Al-Hodiani, “Conditional inclusion dependencies for improving xml data consistency,” *J. Theor. Appl. Inf. Technol.*, vol. 95, no. 17, pp. 4221–4235, 2017.
- [12] 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.
- [13] 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, pp. 365–371, 2016.
- [14] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis, “Conditional functional dependencies for data cleaning,” *Proc. - Int. Conf. Data Eng.*, vol. 33, no. 2, pp. 746–755, 2007.
- [15] W. Fan, L. Bravo, and S. Ma, “Extending Dependencies with Conditions,” *Constraints*, pp. 243–254, 2007.
- [16] W. Fan and F. Geerts, “Foundations of Data Quality Management,” *Synth. Lect. Data Manag.*, vol. 4, no. 5, pp. 1–217, 2012.
- [17] 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.
- [18] M. Karlinger, M. Vincent, and M. Schrefl, “Inclusion dependencies in XML: Extending relational semantics,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 5690 LNCS, no. 09, pp. 23–37, 2009.
- [19] S. Fajt, I. Mlynkova, and M. Necasky, “On mining XML integrity constraints,” in *2011 Sixth International Conference on Digital Information Management*, 2011, pp. 23–29.
- [20] C. Yu and H. V. Jagadish, “XML schema refinement through redundancy detection and normalization,” *VLDB J.*, vol. 17, no. 2, pp. 203–223, 2008.
- [21] M. Arenas, “Normalization Theory for XML,” vol. 35, no. 4, pp. 57–64, 2006.