Discovering Conditional Inclusion Dependencies

Jana Bauckmann
Hasso-Plattner-Institut, Germany
jana.bauckmann@hpi.uni-potsdam.de

Ziawasch Abedjan
Hasso-Plattner-Institut, Germany
ziawasch.abedjan@hpi.uni-potsdam.de

Ulf Leser
Humboldt-Universität zu Berlin, Germany
leser@informatik.hu-berlin.de

Heiko Müller
Intelligent Sensing and Systems Laboratory, Australia
heiko.mueller@csiro.au

Felix Naumann
Hasso-Plattner-Institut, Germany
naumann@hpi.uni-potsdam.de

ABSTRACT

Data dependencies are used to improve the quality of a database schema, to optimize queries, and to ensure consistency in a database. Conditional dependencies have been introduced to analyze and improve data quality. A conditional dependency is a dependency with a limited scope defined by conditions over one or more attributes. Only the matching part of the instance must adhere to the dependency. In this paper we focus on conditional inclusion dependencies (CINDs).

We generalize the definition of CINDs, distinguishing covering and completeness conditions. We present a new use case for such CINDs showing their value for solving complex data quality tasks. Further, we propose efficient algorithms that identify covering and completeness conditions conforming to given quality thresholds. Our algorithms choose not only the condition values but also the condition attributes automatically. Finally, we show that our approach efficiently provides meaningful and helpful results for our use case.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database applications

Keywords

ontology engineering, Linked Data, data mining

1. PROBLEM STATEMENT

Studying data dependencies, or integrity constraints, has a long history in database research. Traditionally, integrity constraints are defined such that all tuples in a table must obey them, and they have mostly been used to preserve consistency in a database and as a hint to the query optimizer. Recently, a weaker form of dependencies, so called *conditional dependencies*, have gained attention, mostly due to their power in analyzing and improving data quality [9]. A conditional dependency is a dependency with a limited scope, where the scope typically is defined by conditions over several attributes. Only those tuples for which these conditions evaluate to true must adhere the dependency. Research has focussed on two types, i. e., conditional functional dependencies (CFDs) [4] and conditional inclusion dependencies (CINDs) [5]. Results have been published on reasoning about consistency and implication [5, 6], validation of known conditional dependencies [6], or detection of conditional functional dependencies [11, 12].

Interestingly, detecting conditional inclusion dependencies has yet received little attention. We present an entirely new case for CINDs, which clearly shows their value for solving complex data quality tasks. To this end we generalize the established definition for CINDs to cover a subtle, yet important difference between different classes of CINDs. We present algorithms to detect all CINDs and evaluate their efficiency.

1.1 Motivating Use Case

Our motivation for studying CINDs comes from the problem of describing links between objects on the web. Consider, as an example, the problem of interlinking representations of persons in the English and German version of DBpedia[3]. Clearly, many persons have both an English and a German description in DBpedia. Relationships between entries in DBpedia are either represented by using the same URI or by "sameAs"-links; we refer to these relationships as *links*. We use the following relational schema to represent information about persons in DBpedia: `person(pid, cent)`, `birthplace(pid, bplace)`, `deathplace(pid, dplace)` with foreign key relationships from `birthplace.pid` to `person.pid` and from `deathplace.pid` to `person.pid`. Each person has an identifier (`pid`; mostly a person's name), and a century of birth (`cent`). The separate relations for place of birth and place of death result from the fact that persons in DBpedia can have several places of birth or death distinguishing for example the country, region, and city of birth or death. Figure 1 shows (part of) the result of the full outer join over relations `person`, `birthplace`, and `deathplace` on the foreign key attributes in the English version of DBpedia (`Person_EN`) and the German version (`Person_DE`).

Links between persons in `Person_EN` and `Person_DE` in Fig. 1 are represented by an identical `pid`. For some persons in `Person_EN` there is no link to `Person_DE` (and vice versa). Having a German person without a link to an English person (and vice versa), two situations are possible: (1) This English

CIKM'12, October 29–November 2, 2012, Maui, HI, USA.

description does not exist; then the lack of the link truly reflects the database. (2) The English description does exist; then the missing link is a data quality problem. Such problems are very common in scenarios where heterogeneous data sets have overlapping domains but no central authority takes care of properly and bi-directionally linking objects. Many examples of such cases can be found in the world of Linked Open Data [14].

The inclusion dependency `Person_EN.pid` $\subseteq$ `Person_DE.pid` holds only for part of `Person_EN`. The goal of discovering CINDs is to identify conditions within `Person_EN` that summarize properties of those persons that have a link to `Person_DE`. In the given example we can observe a condition `deathplace = United States` $\wedge$ `cent = 18`, which can be explained by the large number of European emigrants in the 19th century to the US. Such conditions are a useful tool to identify candidates for missing links. We propose a method to automatically identify conditions for CINDs between sets of interlinked objects.

pid	cent	birthplace	deathplace
Cecil Kellaway	18	South Africa	United States
Mel Sheppard	18	United States	United States
Buddy Roosevelt	18	Meeker, Col.	Meeker, Col.
Sante Gaiardoni	19	-	-

(a) Relation `Person_EN`

pid	cent	birthplace	deathplace
Cecil Kellaway	18	Kapstadt	Los Angeles
Cecil Kellaway	18	Kapstadt	Kalifornien
Cecil Kellaway	18	Kapstadt	United States
Cecil Kellaway	18	Südafrika	Los Angeles
Cecil Kellaway	18	Südafrika	Kalifornien
Cecil Kellaway	18	Südafrika	United States
Mel Sheppard	18	Almonesson_Lake	Queens
Sam Sheppard	19	-	-

(b) Relation `Person_DE`

Figure 1: Selected data about persons from the English (`Person_EN`) and German (`Person_DE`) DBpedia.

1.2 CIND Discovery and Condition Features

We approach the the problem of CIND detection in three steps: (i) detecting an approximate IND, i. e., an IND that is only satisfied by part of the database, (ii) detecting conditions that can turn an approximate IND into a CIND, i. e., conditions that hold in that part of the database that satisfies the approximate IND, and (iii) choosing a (sub-)set of discovered conditions to build the pattern tableau of the CIND. The first step can be solved using detection methods for approximate INDs, such as [16], or it could be manually performed by an expert user. The problem of finding an optimal pattern tableau has been addressed for CFDs in [12]. In this paper we assume approximate INDs to be given and focus on the second step, namely on efficiently detecting "good" conditions that turn given approximate INDs to CINDs. We outline in [2] how the third step can be realized by applying our algorithms to the ideas of [12].

To achieve our goal of identifying good conditions, we formulate desired features of conditions. In the following we reason over single conditions and their features. A condition on a dependent relation R_1 should distinguish tuples of R_1 included in the referenced relation R_2 from tuples not included in R_2. A condition that is satisfied *only* by included tuples is called a *valid* condition. Simply relying on counting the number of tuples that match a condition, however, may not give the desired results. In our example there are multiple tuples for a single person. If we want to find a condition filtering all included persons, should all tuples for this person match the condition or does one matching tuple suffice? Consider the six tuples for `Cecil Kellaway` in `Person_DE`: `Cecil Kellaway` certainly matches condition `deathplace = Los Angeles`. Counting tuples, however, lets this condition look only one-third as good, because it covers only 2 out of 6 tuples. This problem is common when discovering CINDs over relations that are derived by joining relations in a normalized database.

To account for these discrepancies we introduce a new feature to characterize the scope of conditions: We distinguish *covering* conditions for counting objects, e. g., persons; and *completeness* conditions for counting tuples. More general, a covering condition counts *groups* of tuples whose projection on the inclusion attributes is equal.

We are not only interested in valid conditions that perfectly filter only included persons. To be able to propose missing links, we are also interested in "almost valid" conditions with some non-included persons matching the condition. We use precision and recall measures to quantify the quality of a condition, i. e., the degree of its validity, covering, or completeness (Sec. 2). Our use case then requires to find valid and covering conditions with a certain quality.

Discovering valid and covering, or valid and completeness conditions of a given quality for given approximate INDs poses two major challenges: (i) Which (and how many) attributes should be used for the conditions? (ii) Which attribute values should be chosen for the conditions? Within this paper, we propose algorithms that address both of these challenges. Given an approximate IND, our algorithms find all selecting conditions above a given quality threshold for validity and covering (or completeness) without the need to specify the attributes over which the condition is generated.

The contributions of this paper are as follows:

- A novel *use case* for CIND detection that is motivated by the increasing amount of linked open data;
- A *generalization* of CINDs to distinguish covering and completeness conditions for discovering CINDs over denormalized relations;
- Two *algorithms* that efficiently identify valid and covering (or valid and completeness) conditions, while choosing condition attributes and values automatically;
- A thorough *evaluation* of the algorithms using two real-world data sets. Details on algorithms and experiments are provided in [2].

2. CIND DEFINITION

Formally, a CIND is defined by an embedded approximate IND and a pattern tableau representing the conditions. The following definitions are based on [5] and [9] but we chose a different, yet more intuitive formulation. Let R_1, R_2 be relational schemata over a fixed set of attributes $A_1, A_2, \ldots, A_k$. Each attribute A has an associated domain $dom(A)$. We denote instances of R_1 and R_2 by I_1 and I_2, respectively. Each instance I is a set of tuples t such that $t[A] \in dom(A)$ for each attribute $A \in R$. Let X, X_P and Y, Y_P be lists of attributes in R_1 and R_2, respectively. We use $t[X]$ to denote the projection of t onto attributes X.

Def. 1: Approximate IND An approximate IND $R_1[X] \subseteq R_2[Y]$ is an IND that is satisfied for a non-empty subset of tuples in I_1, i. e., $\exists t_1 \in I_1, t_2 \in I_2 : t_1[X] = t_2[Y]$. $\square$

A tuple $t_1 \in I_1$ satisfies $R_1[X] \subseteq R_2[Y]$ if there exists a *referenced tuple* $t_2 \in I_2$ with $t_1[X] = t_2[Y]$. We call attributes X and Y *inclusion attributes*.

Def. 2: Pattern tableau A pattern tableau T_P restricts tuples of R_1 over attributes X_P and tuples of R_2 over attributes Y_P. For each attribute A 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 $t_p \in T_P$ defines a condition. A constant value for $t_p[A]$ restricts a matching tuple's attribute value to this constant, a dash represents an arbitrary attribute value. A tuple $t_1 \in I_1$ *matches* $t_p \in T_P$ ($t_1 \asymp t_p$) if $\forall A \in X_P$: $t_p[A] = (\text{'–'} \vee t_1[A])$. The definition for a tuple $t_2 \in I_2$ matching $t_p \in T_P$ follows analogously over attributes Y_P. The pattern tableau is divided into a left-hand side (with attributes X_P) and a right-hand side (with attributes Y_P). Both sides of the tableau can be left empty specifying no restriction on any attribute of the respective relation. We call attributes X_P and Y_P *condition attributes*.

Def. 3: Conditional inclusion dependency (CIND) A CIND φ: $(R_1[X; X_P] \subseteq R_2[Y; Y_P], T_P)$ consists of the embedded approximate IND $R_1[X] \subseteq R_2[Y]$ and the pattern tableau T_P over attributes X_P and Y_P defining the restrictions. Sets X and X_P are disjoint, as are Y and Y_P. □

Our example CIND is denoted as follows: φ: (`Person_EN[pid; cent, deathplace]` $\subseteq$ `Person_DE[pid;]`, T_P)

T_P:

cent	deathplace
18	United States

A CIND φ holds for a pair of instances I_1 and I_2 if the following two conditions hold:

1. *Selecting condition on* I_1: Let $t_1 \in I_1$ match any tuple $t_p \in T_P$. Then t_1 must satisfy the embedded IND.
2. *Demanding condition on* I_2: Let $t_1 \in I_1$ match any tuple $t_p \in T_P$. Further, let t_1 satisfy the embedded IND with referenced tuple $t_2 \in I_2$, i. e., $t_1[X] = t_2[Y]$. Then t_2 also must match t_p.

Note that the CIND definition treats selecting conditions and demanding conditions separately and asymmetrically regarding validity and completeness.

Given a CIND φ and instances I_1 and I_2. Let I_φ denote the set of tuples from I_1 that satisfy the embedded IND, i. e., $I_\varphi = I_1 \ltimes_{X=Y} I_2$. We refer to I_φ as the set of *included tuples*. We are also interested in groups of included tuples that have equal values in attributes X, e. g., all tuples for `Cecil Kellaway`. Let g_x denote a *group* of tuples in I_1 having value x for $t[X]$, i. e., $g_x = \{t | t \in I_1 \wedge t[X] = x\}$. We call g_x an *included group* if all tuples are included tuples, i. e., $g_x \subseteq I_\varphi$. A group g_x matches a pattern tuple t_p, denoted by $g_x \asymp t_p$, if any of the tuples in g_x matches t_p, i. e., $g_x \asymp t_p \Leftrightarrow \exists t \in g_x : t \asymp t_p$. Let G_1 denote the set of groups in I_1 and G_φ denote the set of included groups. Finally, for a pattern tuple t_p let $I_1[t_p]$ and $G_1[t_p]$ denote the set of tuples from I_1 and the groups in G_1 that match t_p, respectively.

Def. 4: Valid Condition A condition is *valid* if all tuples of I_1 that match t_p also satisfy the embedded IND, i. e., $I_1[t_p] \subseteq I_\varphi$. □

The validity of a condition can be measured by the precision of this condition, i. e., the number of matching *and* included tuples related to the number of all matching tuples. We call t_p γ-*valid* if it has validity greater than threshold γ. We also define a measure for validity based on groups as the number of matching *and* included groups relative to the number of all matching groups. We call a condition γ-valid$_g$ if it has group validity greater than γ.

Def. 5: Completeness condition A condition is *complete* if it matches all included tuples, i. e., $I_\varphi \subseteq I_1[t_p]$. □

The completeness of a condition can be measured as recall of this condition counting the relation's tuples, i. e., the number of matching *and* included tuples related to the number of all included tuples. We call t_p δ-*complete* if it has completeness greater than δ.

Def. 6: Covering Condition A condition is *covering* if it matches all included groups, i. e., $G_\varphi \subseteq G_1[t_p]$. □

The quality of covering conditions can be measured by the recall of these conditions based on the relation's groups, i. e., the number of matching *and* included groups related to the number of all included groups. We call t_p λ-*covering* if it has recall greater than λ.

3. DISCOVERING GENERAL CONDITIONS

We describe algorithms to detect all γ-valid$_g$ and λ-covering conditions without restricting the attributes that should be used. We provide two different approaches: "Conditional INclusion DEpendency REcognition Leveraging deLimited Apriori" (CINDERELLA) uses an Apriori algorithm and is faster, while "Position List Intersection" (PLI) leverages value position lists and consumes less memory. We present the general idea of both approaches for discovering covering selecting conditions and refer to [2] for details and modifications to discover completeness and demanding conditions.

Our algorithms do not rely on the relational representation of the data. Instead, we choose a representation that allows to handle multiple uses of one attribute or predicate for a single group and that holds information if a group is included or non-included: Given an approximate inclusion dependency $R_1[X] \subseteq R_2[Y]$ we first compute the left outer join $R_1 \mid \bowtie_{X=Y} R_2[Y]$ and group the result on the X attributes. Each group is represented by the following items: (i) the left-hand side inclusion attribute(s), e. g., the person identifier, (ii) a right-hand side inclusion indicator with values INCLUDED for included groups or NULL for non-included groups, and (iii) one item for each (attribute : value)-pair for potential condition attributes, i. e., all attributes of the dependent relation apart from the inclusion attributes. In the following we assume this representation for the embedded IND `Person_DE.pid` $\subseteq$ `Person_EN.pid` to describe our algorithms discovering conditions for persons.

3.1 Using Association Rule Mining

We apply association rule mining to identify conditions like "Whose century of birth is 18 and place of death is 'United States' often also is INCLUDED (in the English DBpedia)". There are two challenges: (i) mapping the problem of condition discovery to association rule mining and (ii) improving efficiency based on characteristics of condition discovery. To leverage association rule mining we prepare our baskets in two steps: We use the modified representation of the left outer join result as described above. We must encode

the affiliation of values to their attributes to form basket items. For our example, we want to be able to distinguish the two conditions `birthplace = Los Angeles` and `deathplace = Los Angeles`. Therefore, we prefix each value with an attribute identifier. Using prefixes A to D for our example yields the following basket for Cecil Kellaway: { INCLUDED, A18, BKapstadt, BSüdafrika, CLos_Angeles, CKalifornien, CUnited_States, D"...Schauspieler"@de }. Now we are able to apply an Apriori algorithm to these baskets to find frequent itemsets and derive rules.

The Apriori algorithm [1] finds all frequent itemsets and uses these frequent itemsets to derive association rules. Apriori uses support and confidence of a rule to prune the search space. In our case the covering of a condition corresponds to the support of a rule in the set of included groups, and the validity of a condition corresponds to the confidence of the rule. A frequent itemset then ensures λ-covering conditions, while the rule generation step filters γ-valid$_g$ conditions.

The following observation exposes an optimization possibility for our problem: We need only rules with right-hand side item INCLUDED, because left-hand side items of such rules build the selecting condition. Thus, we can largely reduce the number of itemsets that must be handled and therefore improve the efficiency of the algorithm. Our CINDERELLA algorithm reduces the number of generated frequent itemsets by only considering itemsets that contain item INCLUDED [2].

3.2 Using Position List Intersection

The Position-List-Intersection (PLI) approach uses a position list representation of values that has been used by the algorithm TANE for discovering functional dependencies [15]. While our approach looks for intersections of lists, the partition refinement of TANE is based on the discovery of subset relationships of position lists.

Position lists (or inverted lists) represent each distinct value in an attribute of a given instance by the set of tuple IDs [15] where the value occurs in. Thus, each attribute is associated with a set of position lists – one for each of its distinct values. In our case, positions are group-IDs (e.g., numbers 1-3 for Cecil Kellaway, Mel Sheppard, and Sam Sheppard). For our example the position list for attribute `cent` are `cent`: $18 = \{1, 2\}$ and `cent`: $19 = \{3\}$.

The idea of PLI is twofold: (i) We use a special position list for included groups, in our example *included*$= 1, 2$. (ii) We cross-intersect position lists of attributes to test value combinations (i. e., conditions) for the intersected attributes, e. g., intersect each position list of attribute A with each position list of attribute B. The covering of a condition then corresponds to the ratio of the cardinality of its position list P intersected with *included* to the cardinality of *included* ($|P \cap \text{included}|/|\text{included}|$). The validity of a condition corresponds to ($|P \cap \text{included}|/|P|$). The PLI algorithm [2] recursively processes the powerset lattice *depth-first* by checking all possible combinations that contain a certain condition. It uses the covering threshold for pruning.

4. EVALUATION

We evaluate our algorithms using two real-life data sets: Persons in the English and German DBpedia 3.6 and Wikipedia image data from [13]. We implemented our algorithms in Java6. Data is stored using a commercial DBMS. All experiments were run on a 2x Xeon quad-core server with 16 GB RAM running a 64bit Linux.

DBpedia use case. There are 296,454 persons represented in the English and 175,457 persons in the German DBpedia data set; 74,496 persons are included in both data sets (identified by using the same URI). We mapped the data sets into relations with 13 attributes, including `pid`, `name`, `birthdate`, `birthplace`, `birthcent`, `deathdate`, `deathplace`, `deathcent`, and a description `descr`. The resulting relations contain 474,630 tuples for the English DBpedia, and 280,913 tuples for the German DBpedia with an intersection of 133,208 tuples.

We identify 85 conditions with a γ-validity of above 0.84 (i. e., twice validity of the empty condition) for German DBpedia persons included in the English DBpedia. The two conditions with the largest covering measure are `descr = American actor`[1] (γ-valid$_g$ = 0.91, λ-covering = 0.029) or `descr = American actress` (0.89, 0.024). These conditions are intuitive and hardly surprising. But we also found unforeseen conditions such as `birthcent = 18` $\wedge$ `descr = American politician` (0.94, 0.015) and `birthcent = 19` $\wedge$ `deathplace = Los Angeles` (0.91, 0.010). In [2] we provide a more detailed evaluation and explanation of discovered conditions.

Wikipedia use case. Golab et al. [13] use two tables of Wikipedia to discover conditions for CINDs: table `Image` with attributes `name`, `size`, `width`, `height`, `bits`, `media_type`, `major_mime`, `minor_mime`, `user`, `user_text`, `timestamp`, `sha1` and table `Imagelinks` denoting links from webpages to image files (attributes `il_from` and `il_to`). The authors in [13] assert the embedded IND `image.name` $\subseteq$ `imagelinks.il_to` and build a pattern tableau with completeness conditions of the preselected attributes `bits`, `media_type`, and `user_text`. The main advantage of our approach over [13] is that the condition attributes need not be pre-selected. Here, we use the same dataset to compare the conditions discovered by both approaches.

If we restrict our algorithms to the same attribute set with the same validity threshold of 0.85 and a completeness of at least 0.003, we discover the same conditions as [13]. CINDERELLA runs 23*s* compared to 18*s* reported by [13] (on presumably different hardware). However, our algorithms also discover more detailed conditions, which cannot be found by [13]: E. g. for condition `media_type = AUDIO` there is another condition `media_type = AUDIO` $\wedge$ `bits = 0` with the exact same validity and completeness. These stricter conditions give more insight into the dataset and prevent wrongly generalizing the identified conditions for similar datasets.

Running our algorithms without restricting condition attributes yields even more interesting results: Unexpectedly, attributes `width` and `height` provide conditions with higher completeness than all other attributes. Conditions `width = 200` $\wedge$ `major_mime = image` and `width = 300` $\wedge$ `major_mime = image` both reach a completeness of 0.04, compared to completeness measures between 0.003 and 0.008 of the previous conditions. Conditions `height = 300` and `height = 200` (each with completeness = 0.02), `height = 240` and `width = 240`, (each with completeness = 0.01) also have higher completeness. These conditions are non-trivial: other widths and heights also appear in the dataset with similar frequency. CINDERELLA runs 78 s to identify 188 conditions with γ-valid > 0.85 and δ-complete > 0.008.

In summary, the ability to select the condition attributes automatically led to the discovery of more completeness con-

[1]Note that we provide translated condition values as the actual value is in German.

ditions satisfying the same validity requirements, which in turn enables to build better pattern tableaux. [13] report an overall support of 0.0636, while we discover already *individual* conditions with a completeness of 0.04, corresponding to a support of 0.03. Our top two conditions already yield a tableau with a completeness of 0.0824 (support 0.0641).

Performance of algorithms. Comparing both algorithms, CINDERELLA is less sensitive in runtime to increasing numbers of identified conditions, while PLI is less sensitive in memory consumption. The amount of included tuples is the decisive factor, not so much the size of the entire data set. We refer to [2] for details and a complexity estimation.

5. RELATED WORK

Conditional inclusion dependencies (CINDs) were proposed by Bravo et al. for data cleaning and contextual schema matching [5]. In [5], complexity bounds for reasoning about CINDs and a sound and complete inference system for CINDs are provided. The problem of discovering CINDs from a given database instance, however, is not addressed. De Marchi et al. propose data mining algorithms to discover approximate INDs [16]. Approximate INDs are input to the algorithms presented in this paper. Algorithms for generating pattern tableaux for given INDs are proposed in [8, 13]. The algorithm in [8], however, does not ensure or check validity of conditions. Golab et al. present *Data Miner*, a system for analyzing data quality [13]. The two main differences to our approach are that (1) we do not require the condition attributes to be pre-selected, and (2) we introduce the new concept of covering INDs, which is essential for the type of data and use case that we consider.

The algorithm in [13] extends the one proposed in [12] for generating pattern tableaux for conditional functional dependencies (CFDs), which were introduced in [10]. Algorithms for discovering CFDs are also considered in [7, 11]. In contrast to other approaches, the work in [7] does not assume that the FD is given in advance. Discovering FDs, however, is significantly different from discovering approximate INDs and it therefore is not clear how the algorithms in [7] can be applied to CIND discovery. Fan et al. propose algorithm *CFDMiner* for discovering constant CFDs based on closed itemset mining [11]. A minimal constant CFD is a CFD for which the pattern tableau contains only constant values for the attribute in the right-hand side of the embedded FD. Thus, minimal constant CFDs correspond to association rules with single attribute in their antecedent with confidence 100 %, i. e., to selecting conditions with γ-validity one. Contradiction patterns are also a form of association rules with fixed antecedent [17]. Contradiction patterns were proposed to discover conditions that are frequent within a subset of a database but not frequent within the remainder of the database. The definitions of conflict relevance and conflict potential are similar to our definitions of valid and completeness conditions. Covering conditions, however, cannot be discovered using the algorithms presented in [17].

6. CONCLUSION AND FUTURE WORK

We generalize the definition of CINDs by distinguishing covering and completeness conditions. This distinction is important when discovering CINDs over denormalized relations. To discover CINDs we present algorithms CINDERELLA and PLI. In contrast to existing approaches, both algorithms not only select the condition values but also the condition attributes automatically. CINDERELLA is faster than PLI, but consumes more memory. In our experimental evaluation we identified comprehensible, but unforeseen conditions that highlight characteristics of persons for which there exists a link between the English and German version of DBpedia. We plan to adapt the distinction of covering and completeness conditions to the right-hand side of the pattern tableau.

7. REFERENCES

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proc. of the Int. Conference on Very Large Databases (VLDB)*, 1994.

[2] J. Bauckmann, Z. Abedjan, U. Leser, H. Müller, and F. Naumann. Covering or complete? discovering conditional inclusion dependencies. Technical report, Hasso-Plattner-Institut für Softwaresystemtechnik an der Universität Potsdam, 2012.

[3] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. Dbpedia - a crystallization point for the web of data. *J. Web Sem.*, 7(3):154–165, 2009.

[4] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for data cleaning. In *Proc. of the Int. Conference on Data Engineering (ICDE)*, 2007.

[5] L. Bravo, W. Fan, and S. Ma. Extending dependencies with conditions. In *Proc. of the Int. Conference on Very Large Databases (VLDB)*, 2007.

[6] W. Chen, W. Fan, and S. Ma. Analyses and validation of conditional dependencies with built-in predicates. In *Database and Expert Systems Applications*, 2009.

[7] F. Chiang and R. J. Miller. Discovering data quality rules. *Proc. of the VLDB Endowment*, 1:1166–1177, 2008.

[8] O. Curé. Conditional inclusion dependencies for data cleansing: Discovery and violation detection issues. In *Proc. of the Int. Workshop on Quality in Databases (QDB)*, 2009.

[9] W. Fan. Dependencies revisited for improving data quality. In *Proc. of the Symposium on Principles of Database Systems (PODS)*, 2008.

[10] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for capturing data inconsistencies. *ACM Transactions on Database Systems (TODS)*, 33(2):1–48, 2008.

[11] W. Fan, F. Geerts, J. Li, and M. Xiong. Discovering conditional functional dependencies. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 23(4):683–698, 2011.

[12] L. Golab, H. Karloff, F. Korn, D. Srivastava, and B. Yu. On generating near-optimal tableaux for conditional functional dependencies. *Proc. of the VLDB Endowment*, 1:376–390, 2008.

[13] L. Golab, F. Korn, and D. Srivastava. Efficient and effective analysis of data quality using pattern tableaux. *IEEE Data Engineering Bulletin*, 34(3):26–33, 2011.

[14] H. Halpin, P. Hayes, J. P. McCusker, D. McGuinness, and H. S. Thompson. When owl:sameas isn't the same: An analysis of identity in linked data. In *Proc. of the Int. Semantic Web Conference (ISWC)*, 2010.

[15] Y. Huhtala, J. Kaerkkaeinen, P. Porkka, and H. Toivonen. TANE: an efficient algorithm for discovering functional and approximate dependencies. *The Computer Journal*, 42(2):100–111, 1999.

[16] F. D. Marchi, S. Lopes, and J.-M. Petit. Unary and n-ary inclusion dependency discovery in relational databases. *J. Intell. Inf. Syst.*, 32:53–73, 2009.

[17] H. Müller, U. Leser, and J.-C. Freytag. Mining for patterns in contradictory data. In *Proc. of the SIGMOD Int. Workshop on Information Quality for Information Systems (IQIS)*, 2004.