Efficient and Effective Analysis of Data Quality using Pattern Tableaux

Lukasz Golab, Flip Korn and Divesh Srivastava
AT&T Labs - Research
180 Park Avenue, Florham Park NJ, 07932, USA
{lgolab, flip, divesh}@research.att.com

Abstract

Data Auditor is a system for analyzing data quality via exploring data semantics. Given a user-supplied constraint, such as a functional dependency or an inclusion dependency, the system computes pattern tableaux*, which are concise summaries of subsets of the data that satisfy (or fail) the constraint. The engine of Data Auditor is an efficient algorithm for finding these patterns, which defers expensive computation on patterns until needed during search, thereby pruning wasted effort. We demonstrate the utility of our approach on a variety of data as well as the performance gain from employing this algorithm.*

1 Introduction

Recently, a constraint-based approach has shown promise in detecting and correcting data quality problems; some specific examples of constraints employed include Conditional Functional Dependencies (CFDs) [7], Conditional Inclusion Dependencies (CINDs) [2], and Conditional Sequential Dependencies (CSDs) [10]. A key idea behind this approach is to use *conditioning*, that is, to allow different (but possibly overlapping) subsets of the data to satisfy a supplied constraint locally, rather than forcing the entire data to satisfy the constraint globally. This is especially useful when the data evolve over time or have been integrated from multiple sources and, thus, exhibit heterogeneity with locally shared characteristics.

We have developed a system—Data Auditor—that employs this approach as follows (see [9] for more details). The user first hypothesizes a constraint, either based on expertise and intuition, or one that is automatically derived (algorithms for doing so are outside the scope of this paper). What Data Auditor provides is a platform for testing such hypotheses, performing multidimensional analysis to discover and parsimoniously summarize which portions of the data fit (or fail) the given constraint. That is, users can "try out" a constraint to see if (and where) it holds or fails.

Discovering and summarizing which subsets of the data satisfy a supplied constraint is useful for understanding the semantics of data. This, in turn, is useful not only for making sense of analysis results, but also for detecting quality problems, which may be indicated by violations of these semantics. As a caveat, violations of a user-specified constraint do not always indicate data quality problems but could be naturally occuring (and perhaps interesting) phenomena in the data that were unknown to the user; it is up to the domain expert to make this inference.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

There are two crucial observations to make this approach effective on real world data. The first is that the notions of satisfaction and violation of a constraint should be robust to outliers. For example, a single spurious tuple should not cause a certain portion of the data to be eliminated from consideration. Therefore, a small number of exceptions are allowed before declaring failure of constraint satisfaction; this relaxation is quantified as the *confidence* of a constraint. The other observation is that the reported summary of semantics should not overfit the data. On the one hand, satisfaction of the constraint should be pervasive enough in the data to be considered a legitimate rule. At the same time, the constraint should not be applied to regions of the data having poor quality. Therefore, we do not require all the data to be captured in the summary but only a minimum fraction, denoted by the *support*. This flexibility enables the constraint to "snap" to the portion of best fit.

The mechanism by which the (satisfaction or violation of a) constraint is concisely summarized is a special kind of relation called a *tableau*, which we define in the next section. Briefly, a tableau consists of patterns, each of which lists the attribute values that most (but not necessarily all) satisfying or violating tuples have in common. Effectively and efficiently finding such tableaux is the subject of this paper. Previous work assumed that the attributes used to construct subsets of the data on which to test the given constraint are shared with those specified in the constraint itself [11]; this assumption was made for the purpose of efficiency, as the so-called "on-demand" algorithm [11] for tableau discovery exploited this fact. In this paper, we decouple the conditioning attributes from those in the constraint and discuss a more general on-demand algorithm. As a result, more powerful tableaux analyses are possible and can be done efficiently.

The remainder of this paper is organized as follows. Section 2.1 gives examples of two constraints that are common in Data Auditor. Section 2.2 defines patterns and tableaux. Section 2.3 presents the tableau generation problem and summarizes the on-demand algorithm. Section 2.4 illustrates specification within Data Auditor using the two examples. Section 3 gives a few case studies to demonstrate the utility of generalized pattern tableaux and reports the performance improvements of the on-demand algorithm over straightforward computation. Section 4 discusses related work and Section 5 concludes the paper.

2 Overview of Pattern Tableau Discovery

2.1 Supported Constraints

The supported constraints assert some property between two sets of attributes, X (corresponding to the LHS) and Y (corresponding to the RHS), from either the same or different relations. These attribute sets can be used in a variety of constraints, such as Equality-Generating Dependencies (EGDs) [6] to impose consistency and Tuple-Generating Dependencies (TGDs) [6] to impose completeness. We can define views corresponding to an arbitrary conjunction of atomic formulas on the LHS (for EGDs and TGDs) and the RHS (for TGDs), in which case simple constraints can be defined over these view tables. The requirements for these constraints will become clear in Section 2.3.

A common EGD constraint, which we highlight in this paper, is the functional dependency. Let R be a relation, t_k be a tuple and $t_k[Z]$ be the value(s) of its attribute(s) Z. A functional dependency $X \rightarrow Y$ is said to hold when $\forall i, j$, if $t_i[X] = t_j[X]$ then $t_i[Y] = t_j[Y]$. $X \subseteq R$ is a set of attributes referred to as the *antecedent* and $Y \subseteq R$ is a set of attributes referred to as the *consequent*.

A common TGD constraint is the inclusion dependency $R_1.X \subseteq R_2.Y$, for two relation schemas, R_1 and R_2. It is said to hold when for all tuples r_1 in the domain of R_1, there exists a tuple r_2 in the domain of R_2 with $r_1[X] = r_2[Y]$.

2.2 Pattern Tableaux

Real data are heterogeneous. Hence, in addition to reporting the confidence (i.e., the extent to which the constraint holds) on the entire relation, it is useful to identify parts of the relation that satisfy the constraint (i.e., have high confidence) and parts that violate it (i.e., have low confidence).

Following [7], we use *pattern tableaux* to encode subsets of a relation. Consider a schema $R = A_1, A_2, \ldots, A_k$. A tableau consists of a set of *patterns* over the selected *conditioning attributes* $X_p \subseteq R$, where each attribute in X_p contains a *symbol*. A symbol is either a value in the corresponding attribute's domain or a special "wildcard" symbol '*'. Let $p_i[A_j]$ denote the symbol corresponding to the jth conditioning attribute of the ith pattern, and let $t[A_j]$ be the value of the jth attribute of a tuple t. A tuple t is said to *match* a pattern p_i if, for each A_j in X_p, either $p_i[A_j] =$ '*' or $t[A_j] = p_i[A_j]$. We denote this matching operator as $t[A_j] \asymp p_i[A_j]$. The confidence of a pattern p_i is defined as the confidence of a sub-relation containing only those tuples that match p_i; its support is defined as the relative size of said sub-relation as compared to the whole relation. Note that the pattern consisting of all '*' symbols matches the entire relation.

2.3 Tableau Generation

The input to Data Auditor is:

- a pair of schemas R_1 and R_2 (which optionally can refer to the same relation);
- a constraint involving two sets of attributes, $X \subseteq R_1$ and $Y \subseteq R_2$, encoded as User-Defined Functions (UDFs) for incrementally computing the confidence of this constraint over a set of tuples;
- a set of conditioning attributes $X_p \subseteq R_1$;
- confidence threshold $\hat{c}$ (which is either a lower- or upper-bound on the confidence of each pattern)
- minimum support threshold $\hat{s}$.

The output is a tableau T_p over attributes X_p, each of whose patterns has confidence of at least (or at most[1]) $\hat{c}$, all of whose patterns collectively cover at least $\hat{s}|dom(R_1)|$ tuples, where $dom(R_1)$ denotes the instance of relation R_1. Applying Occam's Razor, the goal is for $|T_p|$ to be as small as possible while satisfying these constraints. This was formulated as an instance of PARTIAL SET COVER in [11], for which a greedy set cover heuristic attempts to produce the smallest possible tableau (having the fewest patterns) matching the largest possible fraction of the relation. Thus, general patterns with wildcards are more likely to be included (provided they have the appropriate confidence) than specific patterns matching a small fraction of the data. Tableau construction for more general constraints extends the *on-demand* algorithm from [11], which was originally proposed to generate tableaux for CFDs where the conditioning attributes, X_p, and FD antecedent attributes, X, are the same (see [11] for a detailed explanation). The only requirement for the generalization in this paper is that the confidence of any pattern can be computed in a single scan after sorting the data lexicographically on $X \cup Y$.

Starting from the all-wildcards pattern, the algorithm traverses the different patterns in top-down fashion, along the way inserting into the tableau patterns that meet the required confidence threshold and match the most tuples that have not already been matched. For each pattern p encountered, its support set (the subset of the tuples from R_1 matching p) is scanned for computing confidence. Therefore, the key is to associate a list of each pattern p's support set in the order visited, so that the confidence of any other pattern contained within p can be generated by scanning this list, rather than re-scanning the tuples in R_1.

An essential part of this algorithm is the computation of confidence for a given pattern. The API for specifying the confidence function of any pattern, given the pattern's support set as input, involves supplying six user-defined functions: `declare`, `init`, `update`, `closeXY`, `closeX` and `output`. These functions are used, respectively, to declare relations and attribute sets used by the constraint; initialize counter variables used in confidence computation; maintain these counters incrementally for each tuple t of R_1; "close" a sub-aggregate

[1]As in [11], we refer to the former as a *hold tableau*, since it summarizes subsets on which the constraint holds, and the latter as a *fail tableau*, which detects subsets on which the constraint fails.

when a new value of $t[XY]$ is encountered; "close" a super-aggregate when a new value of $t[X]$ is encountered; and report the final confidence. The output is a confidence value betwen 0 and 1 for the given pattern.

The choice of conditioning attributes is crucial to obtaining concise and informative tableaux—some attributes may be irrelevant to the constraint at hand. Fortunately, in many cases, useful attributes are self-evident. In future work, we plan to study automatic selection of conditioning attributes in more detail, e.g., by exploiting attribute correlations.

2.4 Examples

Define $S(p, R)$ as the support set of pattern p over R, that is, the set of tuples from relation R matching p:

$$S(p, R) = \{t : t \in dom(R) \wedge t[X_p] \asymp p\}.$$

A Conditional Functional Dependency (CFD) ϕ on R is a pair $(R : X \to Y, T_p)$, where (1) $X \to Y$ is a standard FD, referred to as the *embedded FD*; and (2) T_p is a pattern tableau conditioned over attributes X_p, where for each row $t_p \in T_p$ and each attribute $A \in X_p$, $t_p[A] = a$, for some $a \in dom(A)$, or $t_p[A] =$ '*'. One way to define the confidence of a pattern p over R with respect to the embedded FD is to compute the size of the largest "consistent" subset of $S(p, R)$. Formally, given a CFD $\phi = (R : X \to Y, T_p)$, let $Q(p, R)$ denote the set of tuples from $S(p, R)$ that remains after removing the fewest tuples needed to eliminate all violations. Then the confidence of pattern p is $\frac{|Q(p,R)|}{|S(p,R)|}$.[2] Table 1(a) shows UDFs based on this definition of confidence for the CFD $R : \{A, B\} \to C$ conditioned on $\{A, E, F\}$. Note that we find $|Q(p, R)|$ by partitioning the support set on the antecedent attribute(s) X, and, for each distinct value of X, finding the most frequently occurring value of the consequent attribute(s) Y.

A Conditional Inclusion Dependency (CIND) ψ on R_1 and R_2 is a pair $(R_1.X \subseteq R_2.Y, T_p)$, where (1) $R_1.X \subseteq R_2.Y$ is a standard IND, referred to as the *embedded IND*; and (2) T_p is a pattern tableau conditioned over attributes $R_1.X_p$, where for each row $t_p \in T_p$ and each attribute $A \in R_1.X_p$, $t_p[A] = a$, for some $a \in dom(A)$, or $t_p[A] =$ '*'. Given a CIND $\psi = (R_1.X \subseteq R_2.Y, T_p)$, let $Q(p, R_1)$ denote the tuples from $S(p, R_1)$ having a match in $dom(R_2)$. Then one way to define the confidence of pattern p is $\frac{|Q(p,R_1)|}{|S(p,R_1)|}$. Table 1(b) shows UDFs based on this definition of confidence for the CIND $R_1.B \subseteq R_2.C$ conditioned on $\{A, E, F\}$.

3 Case Studies

Here we consider both CFDs and CINDs where the attributes participating in the dependencies are decoupled from the conditioning attributes used for the tableau. In addition to illustrating the utility of discovering tableaux based on these dependencies (for the purposes of data exploration and data cleaning), we demonstrate the performance improvement that comes from employing the on-demand algorithm.

We used 60K records of sales data from an online retailer containing attributes `type, itemid, title, price, vat, quantity, userid, street, city, region, country, zip`. The first six attributes describe the item being purchased, including the tax rate (vat) and the quantity purchased in each order. The remaining attributes describe the client (keyed by userid) who purchased these items. Using the FD `{title}` $\to$ `{price,vat}` and conditioning attributes `{type,region,country}`, we obtained the tableau in Table 2 with $\hat{c} = 0.99$ (only the first six rows are displayed, yielding 0.379 cumulative support).

Since VAT typically applies to the country level, one might expect each separate country to require a separate row in the tableau. Indeed, there was such a partitioning based on uniformity of VAT (many of the individual countries are not shown in Table 2). An even more interesting characteristic captured in this tableau is that the

[2]This is the definition of confidence used in [11]; note that other definitions of confidence (see [13]) can easily be written as UDFs in our framework.

```
declare()
  R1 :  A,B,C,D,E,F;
  R2 :  A,B,C,D,E,F;
  X : A,B;
  Y : C;
  Xp :  A,E,F;
init()
  ntuples = 0;
  freq = 0;
  maxfreq = 0;
  sumfreq = 0;
update()
  ntuples++;
  freq++;
closeXY()
  if (freq > maxfreq)
    maxfreq = freq;
    freq = 0;
close X()
  sumfreq += maxfreq;
  maxfreq = 0;
output()
  return sumfreq/ntuples;
```

(a) CFD confidence

```
declare()
  R1 :  A,B,E,F;
  R2 :  C;
  X : B;
  Y : C;
  Xp :  A,E,F;
init()
  ntuples = 0;
  sumfreq = 0;


update(Tuple t)
  ntuples++;
  if (t[X] in Y) sumfreq++;
closeXY()
  /* do nothing */


close X()
  /* do nothing */

output()
  return sumfreq/ntuples;
```

(b) CIND confidence

Table 1: UDFs for two different constraints

`type`	`region`	`country`	conf	cumSupp
music	*	GBR	0.99	0.111
book	*	GBR	0.99	0.221
*	TX	*	0.99	0.274
*	CA	*	0.99	0.318
*	NY	*	0.99	0.351
*	PA	*	0.99	0.379

Table 2: Hold Tableau from Retailer data ($\hat{c} = 0.99$)

same title may be shared by different forms of media (and have different prices across the different media), which is the reason why music and book items from Britain (GBR) were separated into separate patterns. Note the absence of the tableau row `(dvd,*,GBR)`, indicating that the FD didn't hold for the only other media type (DVDs). After inspecting the data, this turned out to be due to a greater variety of prices for the same item among DVDs than among other media types. Since, unlike Britain, VATs are not uniform throughout the country but vary per region in the United States, the individual states (listed by volume of purchases) needed to form separate patterns to achieve high enough confidence. Here the effect of common titles among different media types did not play as significant a role, since fewer records in each USA state, compared to all of GBR, resulted in almost no conflicts. We also compared the running time of generating this tableau using the on-demand algorithm against that of a standard greedy set cover algorithm that does not employ our optimizations. The performance improvement from computing on-demand was about two-fold, from 550 milliseconds down to 310 milliseconds.

Using 3,601,434 packet traces of IP network attack traffic obtained from the 1999 ACM KDD Cup, including attributes such as `service`, `protocol`, `flag`, `attack_name`, `bytes`, we postulated the FD `{attack_name}` $\rightarrow$ `{flag}` and conditioned on `{service,protocol,attack_name}`, which generated the tableau in Table 3 with $\hat{c} = 0.99$ and $\hat{s} = 0.7$. This revealed the homogeneity of icmp traffic

service	protocol	attack_name	conf	cumSupp
icmp	*	*	0.99	0.626
*	private	*	1	0.783

Table 3: Hold Tableau from KDDcup data ($\hat{c} = 0.99$)

img_bits	img_media_type	img_user_text	conf	cumSupp
*	*	ProteinBoxBot	0.971	0.0334
*	AUDIO	*	0.884	0.0407
7	*	*	0.88	0.047
5	*	*	0.885	0.0525
6	*	*	0.875	0.0576
*	*	Blofeld of SPECTRE	0.986	0.0609
*	*	Melesse	0.991	0.0636

Table 4: Hold Tableau from Wikipedia data ($\hat{c} = 0.85$)

which, after examining the data corresponding to pattern (icmp, ⋆, ⋆), was apparently dominated with echo requests (ecr_i) and very few other protocols. While some of these packets were normal traffic, more than 99% of these ecr_i requests were used for so-called smurf denial-of-service attacks. Examining the data matching pattern (⋆, private, ⋆) revealed that the private protocol almost always occurred as a result of the neptune SYN flooding attack, which accounted for almost 93% of private traffic. Once again, the running time of on-demand was cut in half over the unoptimized greedy set cover algorithm, from 13 seconds to 6.3 seconds.

For discovering CINDs, we used two tables from Wikipedia data: the Image table, containing 777,828 records of multimedia objects with attributes such as img_name, img_size, img_bits, img_media_type, img_user_text; and the Imagelinks table, containing 15,532,084 records of pages containing those objects, with attributes il_from and il_to (denoting links from webpages to image files, respectively). We asserted the IND {Image.img_name} $\subseteq$ {Imagelinks.il_to} to identify images that appeared on web pages, and generated the tableau in Table 4 with $\hat{c} = 0.85$ (only the first 7 rows are displayed, collectively giving 0.0636 support). This tableau reveals that audio content submitted to Wikipedia generally "made it to press" (appeared on a web page), as did 5-, 6- and 7-bit content (note: almost 95% of the content in Image is 8-bit). It also revealed some high-frequency users whose multimedia content generally made it to press such as ProteinBoxBot, which is a well known bot for creating and amending a large number of Wikipedia pages corresponding to mammalian genes (and by far the largest contributor to Image). A fail tableau with $\hat{c} = 0.2$ (not shown) revealed that content created using MS-Office tools (pdf, xls, etc.) was generally rejected, as well as some high-frequency users who provided much content that was not linked from any page. The performance improvement on this data was not as dramatic (18 seconds for on-demand compared to 19 for unoptimized greedy set cover), perhaps due to the overhead in reading the primary keys into a hash table.

We also used Caltrans automobile traffic data polled every 30 seconds from road sensors over the span of a day. Here the tables are Polls, containing fields time, id, district, sensor_type, route, road_name, and StationsTimes, containing the entire cross-product of 1K stations and 2880 times during the day that should have been polled. We asserted the IND {StationsTimes.time,StationsTimes.id} $\subseteq$ {Polls.time,Polls.id} and generated the 8-

district	sensor_type	route	street_name	conf	cumSupp
5	*	*	*	0.934	0.165
80	*	3	*	0.904	0.264
51	*	*	*	0.907	0.341
80	*	1	*	0.916	0.412
99	*	1	*	0.914	0.48
50	OR	2	*	0.912	0.501
99	OR	*	*	0.952	0.518
113	ML	*	*	0.945	0.534

Table 5: Hold Tableau from RoadTraffic data ($\hat{c} = 0.9$)

district	sensor_type	route	street_name	conf	cumSupp
*	*	*	Iron Point Rd	0	0.004
			...		
*	*	*	Saw Mill Rd	0	0.057
5	OR	*	Elk Grove Blvd	0	0.058
			...		
*	ML	*	15th St	0.097	0.062
			...		

Table 6: Fail Tableau from RoadTraffic data ($\hat{c} = 0.1$)

row tableau in Table 5 with $\hat{c} = 0.9$. It reveals some districts and streets where the measurement quality was best, as well as route and sensor-type combinations. The fail tableau using $\hat{c} = 0.1$ in Table 6 revealed that most of the missing measurements occur along the same street, and gives many examples of these. For some of these streets, it is only certain sensor-types or along certain routes for which the problems occurred. There were also some patterns specifying no street at all but rather that the problem occurred along multiple streets. The on-demand algorithm took 2m7s whereas the unoptimized greedy set cover heuristic algorithm took 6m45s.

4 Related Work

A great deal of work exists on data quality analysis and data cleaning; see, e.g., [15] for a survey. In particular, various integrity constraints have recently been proposed to enforce data quality, including Conditional Functional Dependencies (CFDs) [7], Conditional Inclusion Dependencies (CINDs) [2] and Conditional Sequential Dependencies (CSDs) [10]. The key concept behind these constraints is the notion of conditioning: rather than requiring the constraint to hold over the entire relation, it need only be satisfied over conditioned subsets of the data summarized by a *pattern tableau*.

The problem of automatically discovering tableaux for a CFD, given an embedded FD, was first studied in [11, 3]. [3, 8] considered variations of CFD discovery, using different frameworks and optimization criteria, where the focus was on discovering individual patterns; [5] considered CIND discovery using their framework.

Here we extend the on-demand algorithm originally proposed in [11] to allow for several generalizations including arbitrary conditioning attributes (as well as other constraints besides CFDs). The notion of decoupling conditioning attributes from those used in the constraint was employed in [1] and [2].

Finally, we point out the orthogonal problem of discovering which integrity constraints hold on a given relation (in contrast to our goal to discover a tableau for a known constraint). Discovering FDs has been studied in [12, 13]; discovering CFDs in [3, 8]; and discovering INDs has been studied in [14].

5 Conclusions

We demonstrated the approach for data quality analysis employed in Data Auditor: hypothesizing a constraint and then discovering (hold and fail) tableaux which capture the data semantics (and violations of them) by conditioning on the supplied multidimensional attributes. The generalized on-demand algorithm for efficiently finding such tableaux enables a more powerful analysis than was previously available on large data sets. We are considering several directions for future work, among them UDF query optimization, applying our tableau discovery framework to other types of constraints, and discovering the underlying constraints in addition to constructing tableaux for them.

References

[1] L. Bravo, W. Fan, F. Geerts, and S. Ma. Increasing the Expressivity of Conditional Functional Dependencies without Extra Complexity. *ICDE 2008*, 516-525.

[2] L. Bravo, W. Fan, and S. Ma. Extending dependencies with conditions. *VLDB 2007*, 243-254.

[3] F. Chiang and R. Miller. Discovering data quality rules. *PVLDB*, 1(1):1166-1177, 2008.

[4] G. Cormode, L. Golab, F. Korn, A. McGregor, D. Srivastava, and X. Zhang. Estimating the confidence of conditional functional dependencies. *SIGMOD 2009*, 469–482.

[5] O. Cure. Conditional Inclusion Dependencies for Data Cleansing: Discovery and Violation Detection Issues. *QDB Workshop 2009*.

[6] R. Fagin and M. Vardi. The theory of data dependencies - an overview. *ICALP 1984*, 1-22.

[7] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for capturing data inconsistencies. *TODS*, 33(2):1-48, 2008.

[8] W. Fan, F. Geerts, L. Lakshmanan and M. Xiong. Discovering conditional functional dependencies. *ICDE 2009*, 1231-1234.

[9] L. Golab, H. Karloff, F. Korn, and D. Srivastava. Data Auditor: Exploring Data Semantics using Tableaux *PVLDB*, 3(2): 1641-1644 (2010).

[10] L. Golab, H. Karloff, F. Korn, A. Saha, and D. Srivastava. Sequential dependencies. *PVLDB*, 2(1): 574-585 (2009).

[11] L. Golab, H. Karloff, F. Korn, D. Srivastava, and B. Yu. On generating near-optimal tableaux for conditional functional dependencies. *PVLDB*, 1(1):376-390, 2008.

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

[13] J. Kivinen and H. Mannila. Approximate inference of functional dependencies from relations. *Theor. Comput. Sci.*, 149(1):129-149, 1995.

[14] F. D. Marchi, S. Lopes, and J.-M. Petit. Unary and n-ary inclusion dependency discovery in relational databases. *Journal of Intelligent Information Systems*, 32(1):53–73, 2009.

[15] E. Rahm and H. H. Do. Data cleaning: problems and current approaches. *Data Eng. Bulletin*, 23(4): 3-13, 2000.