Workload-Aware Discovery of Integrity Constraints for Data Cleaning

Eduardo H. M. Pena
supervised by Eduardo Cunha de Almeida
Federal University of Technology - Paraná
eduardopena@utfpr.edu.br

ABSTRACT

Violations of integrity constraints (ICs) usually indicate that data are not consistent with a valid representation of real-world entities, and therefore data are dirty. Recent studies present facilities to make dirty databases consistent with a given set of ICs. Yet their evaluation scenarios assume that there will always be an expert ready to feed ICs to their facilities. Manually designing ICs is burdensome, and it is doomed to fail if we consider the dynamic changes of data and applications. An attractive alternative is the automatic discovery of ICs, but the discovery results are mostly accidental. Our research proposal sticks to the automatic discovery approach, but it leverages information found in the application workload to mining semantically valuable ICs. This paper overviews our research on discovering ICs that are particularly suitable for data cleaning, i.e., ICs that are resilient to data and application evolutions.

PVLDB Reference Format:
. . *PVLDB*, 11 (8): xxxx-yyyy, 2018.
DOI:

1. INTRODUCTION

Data profiling is a complex task that helps to uncover relevant metadata for datasets. Typical examples of metadata include basic statistics (e.g., value distributions), patterns of data values, and integrity constraints (ICs). These kind of information provide valuable insights on large datasets by serving various use-cases, such as data exploration, query optimization, and data cleaning [1]. Our research program is built upon data profiling for data cleaning.

Dirty data remain ubiquitous in enterprises. Industry and academia constantly report the severe impact that low data quality have on businesses [6, 11]. The key question, what are dirty data? has been extensively studied [8], with many studies revisiting ICs to ensure business rules over data [7, 11]. In this context, violation of ICs usually indicates that data are not consistent with a valid representation of real-world entities, and therefore data are dirty. Many approaches follow the idea of detecting and repairing IC violations for data cleaning. The development of such research trend has brought many contributions: reasoning about various ICs, other than functional dependencies (FDs) [9]; practical techniques for ICs violation detection [11]; and methods to repair database errors by enforcing ICs [16]; to name but a few.

Proceedings of the VLDB Endowment, Vol. 11, No. 8
Copyright 2018 VLDB Endowment 2150-8097/18/4.
DOI:

It is crucial that there be effective methods to discover ICs from data. Manually designing ICs is a cost-prohibitive and, worse yet, error-prone task. To do so, domain experts would need to keep up with the ever-evolving semantics of data and application. Moreover, the set of IC candidates is usually too large for human validation, even for small datasets. Indeed, many algorithms for automatic discovering of ICs have been developed [14, 5]. In theory, data cleaning pipelines could use IC discover algorithms to mine ICs from sample data, and then use those ICs as data quality rules to detect and repair errors in the database. However, the recent evaluations of IC-based cleaning tools have only used a manual definition of ICs. Therefore, we believe there are still challenges to overcome before a fully automated IC-based data cleaning pipeline can operate.

Discovering ICs from data might return non-reliable results sets. If the discovery is carried out on dirty data, the discovered ICs are likely dirty themselves. Even if the discovery is carried out on cleaned samples of data, which is hard and expensive to get, most of the discovered ICs are likely accidental. The number of discovered ICs radically increases as the number of attributes in the dataset goes up. For example, datasets with dozens of attributes and a few thousands of records produce result sets with thousands of functional dependencies (FDs) [14]. Although some classes of IC allow for implication analysis to discard redundant ICs (e.g., canonical covers of functional dependencies), the number of discovered ICs which shows no semantic meaning remains huge. Furthermore, the effectiveness of current methods for data repairing is highly influenced by the number of ICs the methods consider. Our goal is mining ICs for data cleaning. If these ICs are compliant with the most current semantics of data and application, they should help to repair the database and spot erroneous data in future loads.

We propose an initial approach to discovering ICs that are relevant for data cleaning. We stick to the automatic IC discovery approach, but to support environments in which both data and application evolve we focus on ICs matching the current queries in the pipe. The intuition here is to use the query workload in the application to identify data spots and structural fragments (e.g., set of attributes and predicates) that are important to users. These assets generate a workload characterization that is semantically relevant for the current application. Our approach uses this characterization to guide the IC discovery process. From the workload characterization, our approach estimates a set of features for ICs. Our vision is a classifier that leverages these features to classify a set of ICs based on their applicability for data cleaning. In this thesis, we aim to answer the following research questions:

- How automatic ICs discovery can serve data cleaning?
- How can database workloads benefit IC discovery?

2. RELATED WORK

Most of the recent approaches to IC-based data cleaning manually define the sets of ICs to evaluate their solutions [18, 16]. Much of these approaches assume that ICs and data do not change very often, and use a set of fixed ICs to repair the database through modification and deletions of inconsistent tuples. A few approaches consider evolving ICs. In this case, the database is repaired after modifications to both records and ICs [18]. More recently, the authors of [17] leverage automatic IC discovery for detecting data errors. However, they use the discovered ICs (they focus on FDs) only to generate questions that guide an interaction with experts. The goal of this interaction is to discover as many FD-related errors in data as possible on a budget of questions.

Most works on IC discovery have focused on attribute dependencies. Liu et al. [12] present a comprehensive review of the topic. Papenbrock et al. [14] have looked into implementation details, experimental evaluation, and comparison of various FD discovery algorithms. The studies on IC discovery usually focus on the algorithmic issues of the process, leaving the applicability of the discovered results aside. To handle the large number of discovered ICs, some approaches score ICs on instance-driven metrics for user validation [12, 5]. Unfortunately, these scoring schemes do not consider the dynamic changes in data and application.

3. MINING RELEVANT ICs

This section describes the main steps we are taking towards answering our research questions.

3.1 Denial Constraints

In the context of this thesis, a database is clean if it is consistent with a given set of ICs. Some types of ICs can express semantic rules that others ICs cannot, or vice versa. Denial constraints (DCs) [5] are known to be a response to this expressiveness issue because they generalize important types of ICs, such as functional dependencies (FDs), conditional FDs, and check constraints. Recent works have introduced methods that automatically repair the violations of DCs in the database [16]; hence, our proposal is also based on DCs.

DCs define sets of predicates that databases must satisfy to prevent attributes from taking combinations of values considered semantically inconsistent. Consider a relational database schema $\mathcal{R}$ and a set of operators $\mathcal{W} : \{=, \neq, <, \leq, >, \geq\}$. A DC [2, 5] has the form $\varphi : \forall t_x, t_y, ... \in r, \neg(P_1 \wedge ... \wedge P_m)$, where $t_x, t_y, ...$ are tuples of an instance of relation r of R, and $R \in \mathcal{R}$. A predicate P_i is a comparison atom with either the form $v_1 w_o v_2$ or $v_1 w_o c$: v_1, v_2 are variables $t_{id}.A_j$, $A_j \in R$, $id \in \{x, y, ...\}$, c is a constant from A_j's domain, and $w_o \in \mathcal{W}$. For example, consider a schema of relation $Employees(Name, Manager, Salary, Bonus)$, $Employees \in \mathcal{R}$. A DC φ can express the constraint "*if two employees are managed by the same person, the one earning a higher salary has a higher bonus*" as follows: $\neg(t_x.Manager = t_y.Manager \wedge t_x.Salary > t_y.Salary \wedge t_x.Bonus < t_y.Bonus)$.

An instance of relation r satisfies a DC φ if at least one predicate of φ is false, for every pair of tuples of r, i.e., the predicates of φ cannot be all true at the same time.

3.2 Proposed Approach

We envision an automated data cleaning system. Figure 1 shows an overview of the expected system's architecture. The system regularly collects query logs from the database server to build workloads $\mathbf{W}$. The characterization for workload $\mathbf{W}$ takes into account query structures and tuples that are selected to produce the query results. The outputs of the workload characterization are data samples and attribute clusters.

The DC profiler takes as input data samples, attribute clusters, and, optionally, user-defined DCs templates. The profiler sets the predicate space (from which DCs are mined) according to the attribute clusters. By relying on this predicate space, our DC discovery algorithm discovers approximate DCs from the samples. The next step is to discard the superfluous DCs and select the relevant ones. It is important to discard superfluous DCs so that data repairing only has to consider DCs that capture the current semantics of the data and applications. In [15], we have studied the use of query workload for automatic selection of FDs. However, selecting DCs is more complex and requires further investigation. We have been evaluating decision tree ensembles for this task. The general idea is to estimate different features for DCs, and then use a classifier to decide whether or not DCs are reliable. Feature extraction is based on structural properties of DCs (e.g., how many predicates they have); and metrics that correlate DCs with the database and query workload. The DCs that are classified as reliable are expected to hold in the database as long as the database manipulations produce semantically clean data. Our implementation of the DC classifier is still under development.

The last task is finding and repairing the tuples that violate at least one constraint from the DC profiling results. We have been using HoloClean [16] for this task. Our current setup only uses the DC-based error detection and error repairing modules of the tool. However, we believe that our system could be adjusted to produce different signals to feed the probabilistic graphical model of HoloClean. For example, deriving features from attribute clusters and samples for the graphical model. We postpone this kind of investigation because we have been currently focusing on the data profiling aspects of the system.

3.3 Discovering DCs

We have first worked on the algorithmic issues of discovering DCs from data. At the beginning of our research program, the only known algorithm for DC discovery was FASTDC [5], for which there was no publicly available implementation. We have implemented FASTDC from scratch, and our results agree with those of the original publication. Next, we briefly describe FASTDC, and the overall improvements we have developed for the algorithm.

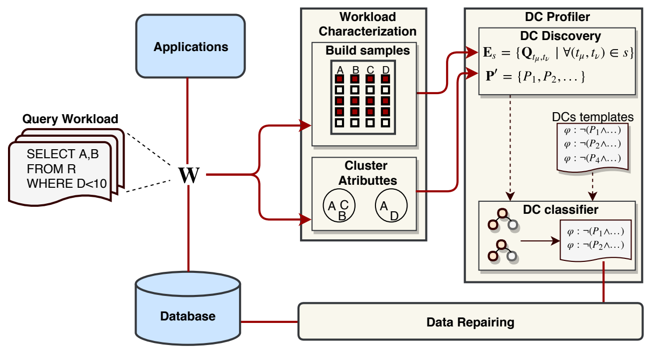
Applications
Query Workload
SELECT A,B
FROM R
WHERE D<10
W
Database
Workload Characterization
Build samples
A B C D
Cluster Atributtes
A C B
A D
DC Profiler
DC Discovery
$\mathbf{E}_s = \{\mathbf{Q}_{t_\mu,t_\nu} \mid \forall(t_\mu,t_\nu) \in s\}$
$\mathbf{P}' = \{P_1, P_2, \dots\}$
DCs templates
$\varphi : \neg(P_1 \wedge \dots)$
$\varphi : \neg(P_2 \wedge \dots)$
$\varphi : \neg(P_4 \wedge \dots)$
DC classifier
$\varphi : \neg(P_1 \wedge \dots)$
$\varphi : \neg(P_2 \wedge \dots)$
Data Repairing

Figure 1: Proposed Data Cleaning Pipeline.

The first step to discover DCs is to set the predicate space $\mathbf{P}$ from which DCs are derived. Experts may define predicates for attributes based on the database structure or use specialized tools for the task, e.g., [19]. The satisfied predicate set $\mathbf{Q}_{t_\mu,t_\nu}$ of an arbitrary pair of tuples $(t_\mu, t_\nu) \in r$ is a subset $\mathbf{Q} \subset \mathbf{P}$ such that for every $P \in \mathbf{Q}$, $P(t_\mu, t_\nu)$ is true. The set of satisfied predicate sets of r is the evidence set $\mathbf{E}_r = \{\mathbf{Q}_{t_\mu,t_\nu} \mid \forall(t_\mu, t_\nu) \in r\}$. Different tuple pairs may return the same predicate set, hence, each $\mathbf{Q} \in \mathbf{E}_r$ is associated with an occurrence counter.

A cover for $\mathbf{E}_r$ is a set of predicates that intersects with every satisfied predicate set of $\mathbf{E}_r$, and it is minimal if none of its subsets equally intersects with $\mathbf{E}_r$. The authors of FASTDC demonstrate that minimal covers of $\mathbf{E}_r$ represent the predicates of minimal DCs [5]. FASTDC uses a depth-first search (DFS) strategy to find minimal covers for $\mathbf{E}_r$.

FASTDC builds the evidence set by evaluating every predicates of the predicate space $\mathbf{P}$ on every pair of tuples of relation instance r. Our improved version of the algorithm, BFASTDC, is a bitwise version of FASTDC that exploits bit-level operations to avoid unnecessary tuple comparisons. BFASTDC builds associations between attribute values and lists of tuple identifiers so that different combinations of these associations indicate which tuple pairs satisfy predicates. To frame evidence sets, BFASTDC operates over auxiliary bit structures that store predicate satisfaction data. This allows our algorithm to use simple logical operations (e.g., conjunctions and disjunctions) to imply the satisfaction of remaining predicates. In addition, BFASTDC can use two modifications described in [5] to discover approximate and constant DCs. These DCs variants let the discovery process to work with data containing errors (e.g., integrated data from multiple sources).

In our experiments, BFASTDC produced considerable improvements on DC discovery performance (up to 24-fold compared to FASTDC). Unfortunately, the number of discovered DCs were unmanageably large, even after selecting only the DCs with high scores for the interestingness dimension described by the authors of FASTDC [5].

Discovering DCs handles large search spaces. Thus, running times for DC discovery increase when more tuples are considered, and dramatically increase when the discovery is set for large predicate spaces. Instead of running BFASTDC over the entire dataset and large predicate spaces, our system focuses the discovery efforts on areas that are relevant for the application workload.

3.4 Workload Characterization

We hypothesize that the information from the application workload (e.g., selection filters in SQL statements) is a powerful asset to narrow the large number of DCs discovered. Query workloads present strong access patterns which, either in horizontal level (individual tuples) or vertical level (individual attributes), points out to specific database regions that are more frequently accessed than others.

Consider a set of user queries $Q = \{q_1, ..., q_m\}$, which is expected to run on the database. For simplicity, we assume there is only one relation in the database. For each query q_i, our system associates the attributes in the operators of the query q_i (e.g., projection and selection) to the attributes of the relation $R(A_1, ..., A_n)$ to define an attribute occurrence matrix (AOM) O as follows:

$$o_{ij} = \begin{cases} 1 & \text{if } q_i \text{ uses attribute } A_j, \\ 0 & \textit{othewise.} \end{cases}$$

Each entry o_{ij} indicates which attributes of relation R is required to answer query q_i.

The AOM model is quite simple but can be enhanced to support dynamic workloads in which the incoming queries are not necessarily identical to each other. We use the approach described in [3], which models workloads as sets of pairs of queries and their weights $\mathbf{W} = \{\langle q_1, w_1\rangle, ..., \langle q_n, w_n\rangle\}$. Each weight w_i is associated with a query q_i, and measures the significance of query q_i in the workload. Thus, AOMs are weighted according to the importance of their queries. The AOMs for each application are straightforward to model if the domain experts are aware of the applications that will run on the database. Furthermore, modern DBMSs commonly provide tools to help with this kind of workload modeling [4].

The authors of [3] describe a probabilistic distribution method to measure the similarity of incoming queries and workload $\mathbf{W}$. Our system uses this method to identify substantial changes in the workload so that it knows when to start profiling DCs again. For each different workload, our system estimates data samples and predicate spaces for which the DC discovery run.

The data samples are expected to represent the data that is most accessed by applications and, therefore, spot data that likely produce most errors. We have been testing three strategies to build samples from queries. The baseline (naive) strategy is building a single sample with every tuple that is used to answer the queries in the workload. The second strategy is building a sample for each fundamental region [3]. They partition the tuples of the relation instance such that each region intersects with the tuples selected by a maximum number of queries. The third strategy is based on a self-pruning splay tree (SPST) index [10]. The SPST keeps the most accessed tuple references near the root of the tree. Thus, our system uses the strategy of [10] to prune the tree and to build samples from the remaining partitions.

Predicate spaces come from AOMs, and restrict the search space of DC discovery for predicates having a high affinity to each other. Our system uses the frequency of attributes in the queries to transform AOMs into attribute affinity matrices (AAMs), in which each entry represents the sum of access frequencies of all queries for that attribute. Further, it uses the graph-based algorithm described in [13] to build a spanning tree from which attribute clusters are derived. These clusters form the predicate spaces as follows. The system defines single and two-tuple predicates on categorical attributes using operators $\{=, \neq\}$; and on numerical attributes using operators $\{=, \neq, <, >, \leq, \geq\}$. It defines predicates involving two different attributes provided that the values of the two attributes are in the same order of magnitude.

BFASTDC runs for every pair of predicate space and data sample. The output of this phase are sets of approximate DCs (i.e., DCs approximately satisfied by the whole dataset). In preliminary experiments, we noticed that many of these DCs were, in fact, exact, or at least had similar structures to many DCs found in exhaustive discovery.

3.5 Features for DCs

Among the discovered DCs, how to choose those that are relevant for data cleaning? To answer that question, we are building a classifier that simulates the experts judging. The main challenge is to form descriptions of "good" DCs. We have included the workload characterization in the discovery process so that our system can estimate features regarding DC's application adherence, and not only estimate features regarding data support.

So far, we have considered the following features for DCs: statistical significance of DCs for the input dataset, as in [5]; the number of predicates [5]; similarity between pairs of DCs, an adaptation of the method described in [15]; similarity between DCs and AAMs, also an adaptation of [15]; and number of DC-related errors in samples and entire dataset.

We have been developing ensembles [20] to combine different decision trees, which are trained upon the above features. For now, the training sets for the learning decision trees are user-defined DCs templates. One could also use the set of DCs currently defined in the database, or manually define acceptable values for the features of the DCs.

4. RESEARCH PLAN

This research program is currently in its second year. The research challenges we will have to face to answer our research questions are as follows. First, we will establish an evaluation scenario in which data, DCs and applications are evolving. So far, we have been running preliminary experiments on datasets that have already been used to evaluate DC discovery [5]. We plan to adapt the evaluation protocols of [18] and [3] for our workload-aware scenario. Then, we will define and detail the metrics that classify DCs as suitable for data cleaning. With these metrics in place, we should be able to devise rules to know when a DC is no longer valid. Such metrics will also help to evaluate the overall system accuracy. Finally, we plan to compare the accuracy and scalability of our system against traditional cleaning solutions.

5. REFERENCES

[1] Z. Abedjan, L. Golab, and F. Naumann. Profiling relational data: A survey. *The VLDB Journal*, 24(4):557–581, Aug. 2015.

[2] L. Bertossi. *Database Repairing and Consistent Query Answering.* Morgan & Claypool Publishers, 2011.

[3] S. Chaudhuri, G. Das, and V. Narasayya. Optimized stratified sampling for approximate query processing. *ACM Trans. Database Syst.*, 32(2), June 2007.

[4] S. Chaudhuri and V. Narasayya. Self-tuning database systems: A decade of progress. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB '07, pages 3–14. VLDB Endowment, 2007.

[5] X. Chu, I. F. Ilyas, and P. Papotti. Discovering denial constraints. *Proc. VLDB Endow.*, 6(13):1498–1509, Aug. 2013.

[6] W. W. Eckerson. Data quality and the bottom line: achieving business success through a commitment to high quality data. Technical report, Data Warehousing Institute, 2002.

[7] W. Fan. Data quality: From theory to practice. *SIGMOD Rec.*, 44(3):7–18, Dec. 2015.

[8] W. Fan and F. Geerts. *Foundations of Data Quality Management.* Morgan & Claypool Publishers, 2012.

[9] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for capturing data inconsistencies. *ACM Trans. Database Syst.*, 33(2):6:1–6:48, June 2008.

[10] P. Holanda and E. C. de Almeida. Spst-index: A self-pruning splay tree index for caching database cracking. In *EDBT, Venice, Italy, March 21-24*, pages 458–461, 2017.

[11] I. F. Ilyas and X. Chu. Trends in cleaning relational data: Consistency and deduplication. *Foundations and Trends in Databases*, 5(4):281–393, 2015.

[12] J. Liu, J. Li, C. Liu, and Y. Chen. Discover dependencies from data - a review. *IEEE TKDE*, 24(2):251–264, Feb. 2012.

[13] S. Navathe, K. Karlapalem, and M. Ra. A mixed partitioning methodology for distributed database design. *Journal of Computer and Software Engineering*, 3(4), 1995.

[14] T. Papenbrock, J. Ehrlich, J. Marten, T. Neubert, J.-P. Rudolph, M. Schönberg, J. Zwiener, and F. Naumann. Functional dependency discovery: An experimental evaluation of seven algorithms. *PVLDB.*, 8(10):1082–1093, June 2015.

[15] E. H. M. Pena and E. C. de Almeida. Uso de instâncias de dados e carga de trabalho para mineração de restrições de integridade. In *XXXII SBBD - Short Papers, Uberlandia, MG, Brazil, October 4-7, 2017.*, pages 312–317, 2017.

[16] T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré. Holoclean: Holistic data repairs with probabilistic inference. *PVLDB Endow.*, 10(11):1190–1201, Aug. 2017.

[17] S. Thirumuruganathan, L. Berti-Equille, M. Ouzzani, J.-A. Quiane-Ruiz, and N. Tang. Uguide: User-guided discovery of fd-detectable errors. In *SIGMOD*, pages 1385–1397, New York, NY, USA, 2017.

[18] M. Volkovs, F. Chiang, J. Szlichta, and R. J. Miller. Continuous data cleaning. In *ICDE*, pages 244–255, March 2014.

[19] M. Zhang, M. Hadjieleftheriou, B. C. Ooi, C. M. Procopiuc, and D. Srivastava. On multi-column foreign key discovery. *PVLDB.*, 3(1-2), Sept. 2010.

[20] Z.-H. Zhou. *Ensemble Methods: Foundations and Algorithms.* Chapman & Hall/CRC, 1st edition, 2012.