CoCo: Interactive Exploration of Conformance Constraints for Data Understanding and Data Cleaning

Anna Fariha
University of Massachusetts Amherst
afariha@cs.umass.edu

Ashish Tiwari
Microsoft
astiwar@microsoft.com

Alexandra Meliou
University of Massachusetts Amherst
ameli@cs.umass.edu

Arjun Radhakrishna
Microsoft
arradha@microsoft.com

Sumit Gulwani
Microsoft
sumitg@microsoft.com

ABSTRACT

Data profiling refers to the task of extracting technical metadata or *profiles* and has numerous applications such as data understanding, validation, integration, and cleaning. While a number of data profiling primitives exist in the literature, most of them are limited to categorical attributes. A few techniques consider numerical attributes; but, they either focus on simple relationships involving a pair of attributes (e.g., correlations) or convert the continuous semantics of numerical attributes to a discrete semantics, which results in information loss. To capture more complex relationships involving the numerical attributes, we developed a new data-profiling primitive called *conformance constraints*, which can model linear *arithmetic* relationships involving multiple *numerical* attributes.

We present CoCo, a system that allows interactive discovery and exploration of **Co**nformance **Co**nstraints for understanding trends involving the numerical attributes of a dataset, with a particular focus on the application of data cleaning. Through a simple interface, CoCo enables the user to guide conformance constraint discovery according to their preferences. The user can examine to what extent a new, possibly dirty, dataset satisfies or violates the discovered conformance constraints. Further, CoCo provides useful suggestions for cleaning dirty data tuples, where the user can interactively alter cell values, and verify by checking change in conformance constraint violation due to the alteration. We demonstrate how CoCo can help in understanding trends in the data and assist the users in interactive data cleaning, using conformance constraints.

1 INTRODUCTION

Data profiling [1] is a key data-management task that involves discovering technical metadata or *profiles* that provide a high-level, informative summary of data. Data profiles encapsulate constraints, patterns, and trends within data and have important applications in data integration, validation, and cleaning. A number of data-profiling primitives exist in the literature such as integrity and denial constraints [3], functional dependencies and its variants (e.g., soft, approximate, relaxed, metric, conditional, pattern) [9], and statistical constraints [11]. However, most of them focus on categorical or text attributes and cannot be trivially extended to (noisy) numerical attributes. To support numerical attributes, existing techniques apply binning or use relational operators to convert the continuous semantics of numerical attributes to a discrete semantics. However, such transformations result in significant information loss. Especially, existing data profiling primitives fall short in capturing the *arithmetic* relationships involving multiple numerical attributes.

Example 1.1. Consider a schema of a flight dataset with the numerical attributes: (1) `departureTime`, (2) `arrivalTime`, (3) `duration`, (4) `distance`, and (5) `delay`. There exist a number of natural constraints involving these attributes that any instance over this schema should (ideally) satisfy. E.g., consider the following two constraints:

(C1) $\texttt{arrivalTime} - \texttt{departureTime} \approx \texttt{duration}$
(C2) $\texttt{AVG\_AIRCRAFT\_SPEED} \times (\texttt{duration} - \texttt{delay}) \approx \texttt{distance}$

The above two constraints can be further used to generate many other constraints. For instance, we can substitute `duration` in C1 with $(\frac{\texttt{distance}}{\texttt{AVG\_AIRCRAFT\_SPEED}} + \texttt{delay})$—obtained from C2—to get a new constraint over `arrivalTime`, `departureTime`, `distance` and `delay`.

Conformance constraints. The constraints in Example 1.1 encode linear arithmetic relationships involving the numerical attributes, which can generally be expressed using the following template:

$$\texttt{LOWER\_BOUND} \leq \sum_i w_i \texttt{A}_i \leq \texttt{UPPER\_BOUND}$$

Here, `LOWER_BOUND` and `UPPER_BOUND` are numerical constants, $\texttt{A}_i$ denotes the i^{th} attribute, and w_i denotes the corresponding numerical coefficient (relative weight). Unfortunately, no existing data profiling primitives is expressive enough to capture these linear arithmetic constraints. To model linear dependencies across numerical attributes within a noisy dataset, we developed *conformance constraints* [8], which can model constraints in the above template.

Automatic discovery of conformance constraints. Traditionally, integrity constraints are specified along with the schema to keep the "integrity" of new data over that schema, e.g., to prevent erroneous tuple insertion. Like other data profiles, conformance constraints can also be specified during schema design. However, it is difficult to come up with the right set of conformance constraints manually. First, real-world data is often noisy and pre-sepcified constraints can be too strict, resulting in unwanted conservativeness during future data operations. Second, figuring out the right set of conformance constraints requires complete understanding of the domain and semantics of each attribute (e.g., `duration` includes both the flight time and the `delay`). Third, finding the coefficients manually is tedious: it requires knowledge of the measurement units of the attributes (e.g., `distance` is in miles). Thus, it is preferable to have a mechanism for automatic discovery of conformance constraints from a given dataset, which we developed in a prior work [8].

Interactive exploration of conformance constraints. A shortcoming of our approach for conformance constraint discovery is that it prioritizes *effectiveness* (finds the strongest conformance constraints)

over *interpretability* (may involve a large number of numerical attributes). For example, if there exists a correlation between the two constraints of Example 1.1 over the data, i.e., all tuples tend to incur similar violation scores against both, then they can be merged to derive a *stronger* constraint. However, such an increase in strength comes at the cost of interpretability, as a conformance constraint that involves too many numerical attributes is less interpretable. To address this issue, CoCo allows the user to tune the parameters for conformance constraint discovery: they can specify the attributes that are of interest and the maximum number of attributes they prefer within the conformance constraints. While this might produce suboptimal constraints, it nonetheless is valuable because it gives users more control and confidence. In particular, CoCo displays the strength of each discovered conformance constraint and also shows top 15 most violating tuples, which helps the user judge the effectiveness of the constraints.

Conformance constraints for data cleaning. An obvious application of conformance constraints is data cleaning. The idea is to first learn conformance constraints over a clean (reference) dataset and then consult the learned constraints for data cleaning. Specifically, violation of the learned constraints by a new tuple indicates that the tuple may be dirty. Furthermore, the closed form expression of conformance constraints also allows us to provide suggestions regarding valid values for each cell (Figure 1), which can guide the user throughout the data cleaning process. CoCo provides an interactive data cleaning solution where the user can edit a cell within a tuple and gets immediate feedback about the corresponding change in constraint violation: reduction or removal of constraint violation confirm a correct cleaning operation (Figure 2).

Related work. We developed complete algorithms for conformance constraint discovery and demonstrated their effectiveness in trusted machine learning and data drift quantification in a prior work [8]. To interpret conformance constraints in the context of trusted machine learning, we previously demonstrated ExTuNe [7], which blames data attributes for causing a tuple's nonconformance, guided by conformance constraints. CoCo significantly differs from ExTuNe, in three ways: (1) Unlike ExTuNe, CoCo allows the user to guide conformance constraint discovery and explore the discovered constraints directly. (2) CoCo associates constraint violation with the constraints themselves and not with the data attributes. (3) CoCo is focused on data cleaning, while ExTuNe's focus was trusted machine learning. In summary, ExTuNe was a causal-intervention-centric system built on top of conformance constraints, while CoCo is a demonstration of the discovery and implication of conformance constraints and their application in interactive data cleaning.

Integrity constraints and functional dependencies have long been used for error detection and data cleaning [2, 5, 6]. Holistic data cleaning [4] provides a unified framework to allow different types of user-provided constraints for data cleaning. Instead of relying on user-provided rules, a practical idea is to automatically discover them from clean data. ANMAT [10] exploits automatically discovered pattern functional dependencies for error detection, but it is limited to text attributes. In summary, none of the existing efforts in data cleaning consider automatically generated constraints involving linear arithmetic expressions over numerical attributes, which is the primary focus of conformance-constraint-driven data cleaning.

In our demonstration, participants will observe how CoCo discovers interpretable conformance constraints, based on their preferences, and experience how conformance constraint violation can facilitate interactive data cleaning. We proceed to describe the solution sketch and then provide an outline of our demonstration.

2 SOLUTION SKETCH

Conformance constraint discovery. The core component of CoCo is a discovery engine for conformance constraints. Conformance constraints enforce that certain projections of the data tuples stay within certain bounds. A projection $\vec{P}$ is simply the sum over a weighted linear combination of the numerical attributes ($\sum_i w_i \mathsf{A}_i$). Given a projection $\vec{P}$, we can compute the bounds of the corresponding conformance constraint by evaluating $\vec{P}$ over the dataset D. In our work, we use the following formulas to compute the bounds, where μ and σ denote mean and standard deviation, respectively:

$$\texttt{LOWER\_BOUND} = \mu(\vec{P}(D)) - 4 \times \sigma(\vec{P}(D))$$
$$\texttt{UPPER\_BOUND} = \mu(\vec{P}(D)) + 4 \times \sigma(\vec{P}(D))$$

However, the key task here is to find a set of good projections that result in strong conformance constraints. To this end, we apply our prior work on conformance constraint discovery [8], which is based on the principles of principal component analysis (PCA). At a high level, the key idea is to find projections that incur *low variance* over the reference data, as this, intuitively, implies that the data shows an "almost constant" trend along that low-variance projection. PCA generates a set of projections (principal components) over a given data, where the first principal component captures the maximum variance and the last principal component captures the least variance of the data. While traditionally, only high-variance principal components have been used to reduce data dimensionality with the aim of reducing reconstruction error, our key idea is that we can use the "by-product" of PCA—the low-variance principal components—to construct strong conformance constraints.

Strength of conformance constraints. A projection with very low variance yields a strong constraint, as "low variance" implies "almost constant" value for a projection, i.e., a steady trend. In contrast, a high-variance projection does not yield a useful constraint. Therefore, we use inverse of standard deviation (square root of variance) of a projection to denote the "strength" of the corresponding conformance constraint: lower the variance, stronger the constraint. We further normalize the strength using the conversion $\lambda\sigma : \frac{1}{\ln(e+\sigma)}$.

Interpretable conformance constraints. Involving all attributes during learning yields the strongest set of conformance constraints. However, as discussed before, it results in poor interpretability. CoCo allows the user to specify a parameter K that denotes the maximum number of attributes preferred within a conformance constraint. Additionally, CoCo allows the user to tune a second parameter $\mathcal{A}$ which denotes a set of attributes over which conformance constraint discovery should be limited. With K and $\mathcal{A}$ specified by the user, CoCo learns constraints on different vertical partitions of the reference dataset, with each partition limited to a subset of K attributes from $\mathcal{A}$. Although, theoretically, this results in combinatorial explosion, the set $\mathcal{A}$ is expected to be small and the value K must be small (≤ 5) for ensuring interpretability. Therefore, in practice, such a runtime complexity is acceptable.

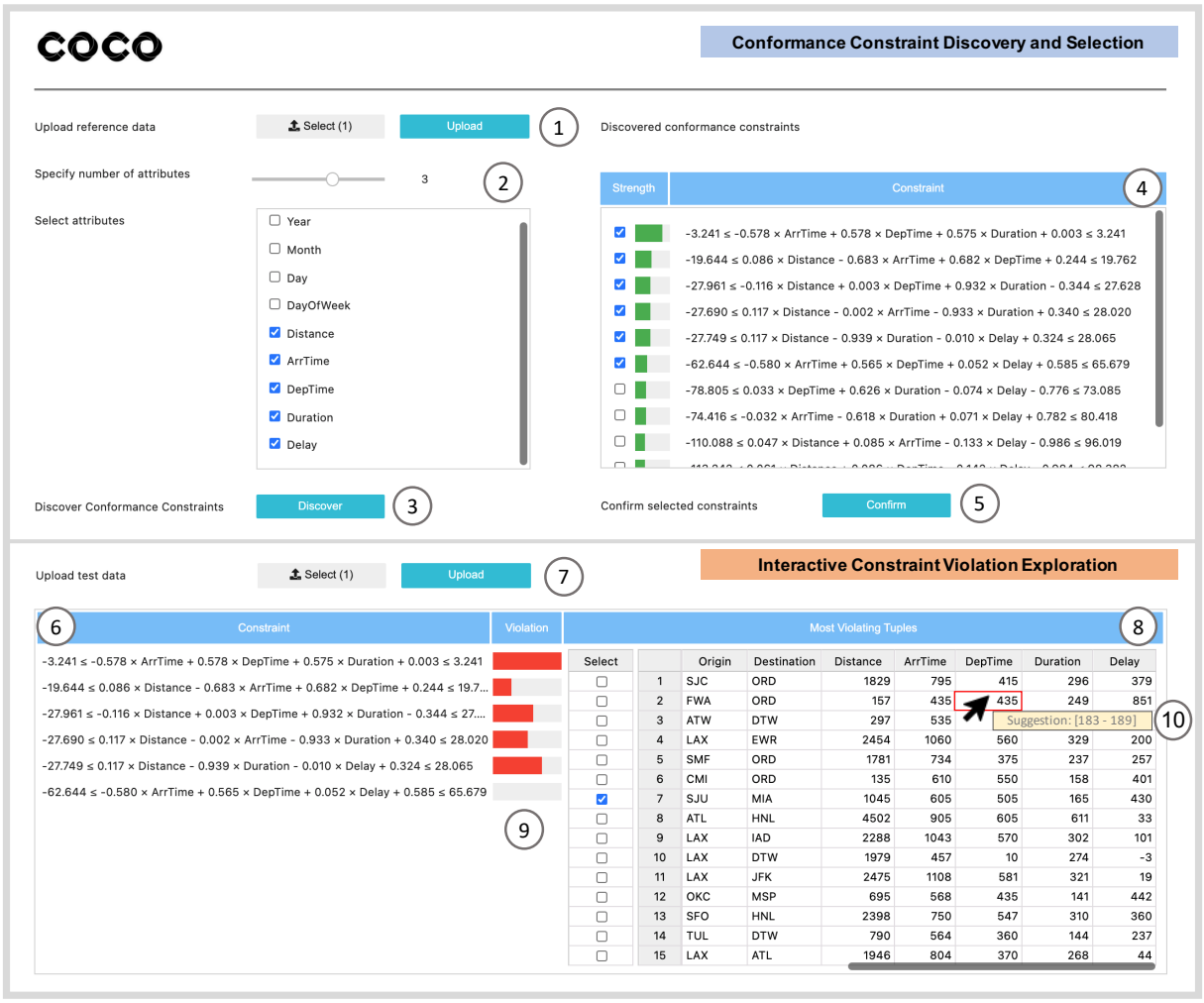

Figure 1: The CoCo demo: ① upload reference (clean) data, ② select relevant attributes and specify the maximum number of attributes desired within the conformance constraints, ③ discover conformance constraints, ④ view the discovered constraints along with their strengths, ⑤ select a subset of conformance constraints for further exploration, ⑥ view the selected conformance constraints, ⑦ upload test (unclean) data, ⑧ view top 15 most violating tuples and select a tuple for checking its violations, ⑨ view constraint-wise violations for the selected tuple, ⑩ hover on a cell to get suggestion on how to alter its value to satisfy the conformance constraints.

CoCo preprocesses the discovered conformance constraints before presenting them to the user. The preprocessing involves (1) removing attributes that are associated with very small weights within a constraint, as this improvs interpretability, and (2) removing redundant constraints that involve the same attributes (and, thus, are equivalent) by keeping only one of the redundant constraints. E.g., the constraints $-2 \leq X+Y \leq 2$ and $-4 \leq 2X+2Y \leq 4$ are equivalent and keeping only one of them is sufficient.

Computing constraint violation. A *violation function* computes how much a tuple t violates a conformance constraint involving projection $\vec{P}$. Specifically, it measures how many standard deviations away $\vec{P}(t)$ is from the bounds of the conformance constraint involving $\vec{P}$:

$$\texttt{violation}\,(t,\ \texttt{lb} \leq \vec{P} \leq \texttt{ub}) = \begin{cases} 0, & \text{if } \texttt{lb} \leq \vec{P}(t) \leq \texttt{ub} \\ \frac{\vec{P}(t)-\texttt{ub}}{\sigma(\vec{P}(D))}, & \text{if } \vec{P}(t) > \texttt{ub} \\ \frac{\texttt{lb}-\vec{P}(t)}{\sigma(\vec{P}(D))}, & \text{if } \vec{P}(t) < \texttt{lb} \end{cases}$$

To aggregate violation scores over a set of constraints, we compute a weighted sum over all constraint violations, where weights are proportional to the strength of the constraint. Finally, we normalize the violation score using the monotonic function $\lambda z : 1-e^{-z}$.

Generating suggestions for data cleaning. For data cleaning, we use a reasonably clean dataset as a reference data. Discovery of conformance constraints requires only a small amount of data that are reasonably clean (number of tuples should be more than the number of attributes for PCA to work). However, our technique for conformance constraint discovery is robust to uniformly distributed outliers across all projections, and straightforward modification in bound computation (tightening) can adjust to noisy data. For a tuple t and a conformance constraint C, assuming all of t's attributes, except the j^{th} attribute, are correct, C can provide us a range of acceptable values for the j^{th} attribute of t that will satisfy C. When we have a set of such constraints C_1, C_2, …, we can generate a

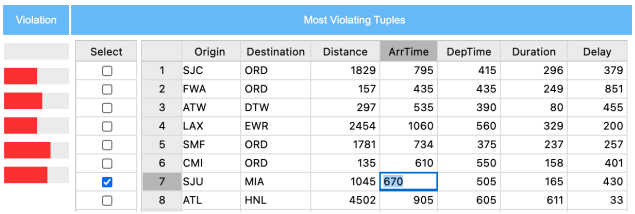

Figure 2: Interactive data cleaning: the user edits a cell in-place and views changes in constraint violation. Changing `ArrTime` from 605 to 670 for the 7 th tuple reduces its violation against the first constraint, but increases violation against the 6th constraint.

range of valid values for each C_i and take their intersection to find a range that will satisfy *all* of the constraints. Using this mechanism, CoCo generates suggestion on how to alter value of a single cell of a tuple to "fix" it. However, we note that when multiple attributes are incorrect, such a suggestion may not be helpful.

3 DEMONSTRATION

We will demonstrate CoCo on a real-world airlines dataset.[1] The dataset contains information about flights over 14 attributes including origin, destination, departure time, arrival time, durtation, distance, and delay. We will use a subset of this dataset that is reasonably clean as a reference dataset.[2] We expect that most participants will be familiar with this data domain and will be able to correctly interpret the conformance constraints that CoCo discovers. Figure 1 shows a screenshot of CoCo's graphical user interface. The top panel is for conformance constraint discovery and selection, and the bottom panel serves the purpose of interactive exploration of constraint violations by data tuples and data cleaning. During the demonstration, we will guide the participants through ten steps. We have annotated each step with a circle in Figure 1.

Step ① (Uploading reference data) First, the user uploads a reference dataset, over which CoCo will learn conformance constraints. Ideally, tuples within the reference dataset should be mostly clean. For our guided scenario, the user uploads a clean subset of the airlines dataset as the reference data.

Step ② (Parameter tuning) Next, the user tunes two parameters for conformance constraints: (1) The maximum number of attributes (K) that can appear in any conformance constraint, which can be specified through a slider. A small value for this parameter yields more interpretable conformance constraints that involve fewer attributes. However, this might compromise the effectiveness of the discovered constraints. For our scenario, the user sets the value of K to be 3. (2) A set of attributes $\mathcal{A}$ over which conformance constraints should be learned. The user selects the attributes {`distance`, `arrivalTime`, `departureTime`, `duration`, and `delay`}, as they deem these attributes as most relevant.

Step ③ (Conformance constraint discovery) The user requests CoCo to discover conformance constraints based on the specified parameters K and $\mathcal{A}$. CoCo learns accordingly and shows a progress bar to keep the user informed about its progress during conformance constraint discovery.

Step ④ (Viewing constraints for data understanding) CoCo presents the discovered conformance constraints to the user, along with their strengths, from which the user can gain insights about the dataset. For example, the first conformance constraint is equivalent to the constraint C1 of Example 1.1. The green bars indicate the strengths of the constraints.

Steps ⑤ - ⑥ (Constraint selection) CoCo can produce many constraints and not all of them are useful for the task at hand. Hence, CoCo lets the user choose a subset of the discovered constraints that they deem useful. Once the user confirms their selection, the selected constraints are shown in the left side of the bottom panel.

Step ⑦ (Uploading test data) The user now uploads a new, potentially dirty, test data for observing to what extent its tuples satisfy or violate the selected conformance constraints. For our guided scenario, the user uploads the dataset containing all flights.

Step ⑧ (Viewing and selecting tuples) CoCo presents the top 15 most violating tuples within the test data to the user, along with their overall violation scores (not shown in Figure 1). The user selects the 7th tuple to dig deeper.

Step ⑨ (Viewing constraint-wise violation) In this step, the user views breakdown of constraint-wise violation by the selected tuple. The length of a red bar denotes the amount of violation against the corresponding conformance constraint.

Step ⑩ (Interactive data cleaning) The last step is designed to facilitate interactive data cleaning. Upon viewing the violations by a tuple, the user may want to edit a cell. Once the user hovers on a cell, CoCo automatically provides suggestion on how to alter its value to satisfy the selected conformance constraints using the mechanism described in Section 2. Once the user edits a cell, CoCo instantly updates the red violation bars according to the altered tuple. Figure 2 shows a snapshot of such an operation.

Demonstration engagement. After our guided demonstration, participants will be able to plug their own datasets into CoCo. Through the demonstration, we will showcase how CoCo can effectively learn useful conformance constraints according to the user's preference. The key takeaway is that conformance constraints provide a natural way for data understanding and interactive data cleaning.

[1] Airlines dataset: http://kt.ijs.si/elena_ikonomovska/data.html
[2] We generated this clean dataset through manual inspection.

REFERENCES

[1] Z. Abedjan, L. Golab, and F. Naumann. Profiling relational data: a survey. *VLDB J.*, 24(4):557–581, 2015.
[2] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for data cleaning. In *ICDE*, pages 746–755, 2007.
[3] X. Chu, I. F. Ilyas, and P. Papotti. Discovering denial constraints. *Proceedings of the VLDB Endowment*, 6(13):1498–1509, 2013.
[4] X. Chu, I. F. Ilyas, and P. Papotti. Holistic data cleaning: Putting violations into context. In *ICDE*, pages 458–469, 2013.
[5] W. Fan, F. Geerts, and X. Jia. A revival of integrity constraints for data cleaning. *PVLDB*, 1(2):1522–1523, 2008.
[6] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for capturing data inconsistencies. *TODS*, 33(2):6:1–6:48, 2008.
[7] A. Fariha, A. Tiwari, A. Radhakrishna, and S. Gulwani. ExTuNe: Explaining tuple non-conformance. In *SIGMOD*, pages 2741–2744, 2020.
[8] A. Fariha, A. Tiwari, A. Radhakrishna, S. Gulwani, and A. Meliou. Conformance constraint discovery: Measuring trust in data-driven systems. In *SIGMOD*, 2021. https://people.cs.umass.edu/afariha/ConformanceConstraints.pdf.
[9] 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, 2015.
[10] A. A. Qahtan, N. Tang, M. Ouzzani, Y. Cao, and M. Stonebraker. ANMAT: automatic knowledge discovery and error detection through pattern functional dependencies. In *SIGMOD*, pages 1977–1980, 2019.
[11] J. N. Yan, O. Schulte, M. Zhang, J. Wang, and R. Cheng. SCODED: statistical constraint oriented data error detection. In *SIGMOD*, pages 845–860, 2020.