CODED: Column-Oriented Data Error Detection with Statistical Constraints

Jing Nathan Yan[†], Oliver Schulte[‡], Jiannan Wang[‡], Reynold Cheng[†]
[‡]Simon Fraser University [†]The University of Hong Kong
[‡]{jnwang, oschulte}@sfu.ca [†]{jyan,ckcheng}@cs.hku.hk

ABSTRACT

Traditional integrity constraints (e.g., functional dependencies and denial constraints) detect data errors by treating each row as an entity and comparing two selected entities to determine whether a constraint is violated or not. In this paper, we introduce a class of column-oriented constraints, called statistical constraints (SCs), which treat each column as a random variable and enforce an independence or dependence relationship between two random variables. We find that SCs are expressive, and work harmoniously with downstream statistical modeling. We develop the CODED error detection system based on the proposed SCs, which solves a number of challenging problems. First, we provide a novel algorithm for checking whether a given set of SCs is consistent. Second, for a given dataset, an SC does not hold absolutely, but only to a certain degree. We show how well-established statistical metrics can be used to quantify the degree to which an SC holds or fails in a given dataset. Third, we propose an error-drill-down framework, and devise efficient algorithms that identify the top-k records that contribute the most to the violation of an SC. Experiments on both synthetic and real-world data illustrate how SCs can be applied for error detection, and provide evidence that CODED performs better than state-of-the-art approaches.

1. INTRODUCTION

Error detection is the problem of detecting erroneous (inconsistent, inaccurate, incorrect) values in a database. It has been extensively studied in the past but is still far from being solved [3]. Data errors can cause many serious issues, such as wrong decision making and biased machine-learning models. In 2016, IBM estimated that poor data quality costs the U.S. economy around $3 trillion per year [1]. As more and more companies assign data a central place in their business, error detection will become an increasingly serious problem.

In the database community, error detection is mostly solved by detecting the violations of integrity constraints (ICs) [24]. Intuitively, a user first specifies an IC that describes what the data should look like, and then detects which parts of the data violate the constraint. Take functional dependencies (FDs) as an example. Consider a restaurant database with Name, Address, Zipcode, and City attributes. Suppose the given FD is Zipcode → City, which means that if two records have the same Zipcode, then they must have the same City. By comparing each pair of records in the database, if two records have the same Zipcode but different City, then they violate the constraint, thus an error is detected.

Table 1: A comparison between ICs and SCs.

	Data Model	Violation Definition
IC	Row as an entity	Between two (or a few) entities
SC	Column as a random variable	Between two (or a few) random variables

Traditional integrity constraints, such as functional dependencies (FDs) [7], conditional functional dependencies (CFDs) [8], and denial constraints (DCs) [12], are *row-oriented* constraints. That is, they model each row as an entity and view a data table as a collection of entities. An IC is used to enforce a relationship among two (or a few) entities. For the above example, each row represents an entity (i.e., restaurant) and the FD enforces a relationship between *two* restaurants. That is, a violation is detected by comparing two restaurants only.

The statistics community takes a different data model, which treats each column as a random variable, and views a data table as a sample drawn from the joint distribution for these random variables. The joint distribution gives rise to independence or dependence relationships among the random variables. For example, consider a Car database with Model, Fuel Efficiency, Price, and Color attributes. This database represents a data sample from the joint distribution for the four random variables. Some example independence/dependence relationships are that Fuel Efficiency and Price are dependent, and Color and Fuel Efficiency are independent. These constraints are *column-oriented* because, for example, they treat Fuel Efficiency and Price as two random variables and a violation is detected based on their entire distributions.

We call this kind of constraint *Statistical Constraints* (SCs). SCs are preferable to ICs in a number of situations. First, a user may sometimes know that a set of columns are (in)dependent but it may not be easy to express such relationships using ICs. For example, it is not easy to come up with a set of ICs to express the dependence relationship between Fuel Efficiency and Price. Instead, she can easily specify an SC to enforce this relationship.

Second, SCs enable error detection and statistical modeling, the two separate components in a data analysis pipeline, to complement each other smoothly. This is due to the fact that they use the same data model (i.e., modeling each column as a random variable). Error detection tries to highlight unreasonable relationships among random variables caused by dirty data; statistical modeling seeks to learn correct relationships among random variables from

ID	Price	Model
a1	31000	BMW X1
a2	32000	BMW X1
a3	33000	BMW X1
a4	34000	BMW X1
b1	18000	Toyota Prius
b2	19500	Toyota Prius
b3	18300	Toyota Prius
b4	18300	Toyota Prius

*(a) SC_1 = "**ID** and **Price** are independent given **Model**"*

ID	Model	Color
a1	BMW X1	White
a2	BMW X1	Black
a3	BMW X1	White
a4	BMW X1	Black
b1	Toyota Prius	**White**
b2	Toyota Prius	**White**
b3	Toyota Prius	**White**
b4	Toyota Prius	**White**

(b) *SC_2 = "**Model** and **Color** are independent"*

Figure 1: Examples of data errors detected by SCs

clean data. For example, suppose a statistical model learns that *Fuel Efficiency and Price are independent*. If a user thinks that this relationship is unreasonable, she can construct an SC like *Fuel Efficiency and Price are dependent*, and then enforce the constraint on the data to detect errors.

Data errors, especially systematic errors across a set of records, often affect the independence or dependence relationship among columns. Figure 1 shows two examples. In Figure 1(a), consider SC_1 = *"ID and Price are independent given Model"*. Suppose the data is integrated from two sources, "bmw.com" and "toyota.com". However, in "bmw.com", the data is sorted by Price. When merging them together, ID and Price become highly dependent given Model = "BMW X1"[1], which violates SC_1. In Figure 1(b), consider SC_2 = *"Model and Color are independent"*. Suppose the original data has some missing colors (e.g., the colors of all Toyota cars are missing), and one replaces them with "White" (highlighted in the figure). The consequence is that Model and Color become highly dependent. That is, if the Model is "Toyota Prius", the Color must be "White". Thus, the data violates SC_2

In this paper, we consider four kinds of SCs: (1) A and B are dependent; (2) A and B are independent; (3) A and B are dependent given C; (4) A and B are independent given C, where A, B, and C are three columns. There are certainly some other kinds of SCs that are more complex (e.g., A and BC are independent given DE). We choose simple types because they are easy to be interpreted and provided, and they are useful to detect errors in many scenarios (e.g., see the examples in Figure 1). The expressive power of these SCs is sufficient to evaluate the viability and effectiveness of our statistical approach. Furthermore, even restricted SCs give rise to challenging research problems, as follows.

Consistency and Implication. The consistency and implication problems are fundamental to constraint-based data cleaning approaches [39, 8, 13]. Given a set of SCs, the consistency problem is to determine whether they have conflicts; the implication problem is to determine whether they imply another SC. The key to solve these problems is to develop a sound and complete inference system. For row-oriented constraints, the inference systems are mainly derived from Armstrong's Axioms [4]. Since SCs are column-oriented, Armstrong's Axioms are not applicable, thus we develop a new inference system from the well-known Graphoid Axioms [18]. Our inference system is proved to be sound and conjectured to be complete. We use it to develop efficient consistency-checking and implication algorithms, with the time complexity of $\mathcal{O}(n^2)$, where n is the number of given SCs.

Error Detection. Given a (dirty) dataset and an SC, we need to determine whether the dataset violates the SC or not. We solve this problem using Hypothesis Testing. Note that although hypothesis testing is a well-studied topic, to the best of our knowledge, none of existing work has applied it in an error detection setting. We further study how to drill down into erroneous values rather than just return erroneous columns. We propose two effective top-k algorithms, which aim to return the top-k most likely erroneous values. For example, suppose a user knows that Model and Color should be independent, but hypothesis testing shows that they are highly dependent in the data. The user may want to know which values in the Model and Color columns cause the correlation. The top-k algorithms can identify the top-k problematic records for the user.

To summarize, we make the following contributions:

- We formally define SCs, and present CODED, an SC-based error-detection system.
- We propose a new inference system for SCs, which is proved to be sound and conjectured to be complete. We develop efficient consistency-checking and implication algorithms based on the new inference system.
- We apply Hypothesis Testing to detect erroneous columns. We further formalize the problem of drilling down into erroneous values, and propose two top-k algorithms to solve it.
- We conduct extensive experiments on real-world datasets with both simulated errors and real errors. The results demonstrate the superiority of our approaches over the state-of-the-art approaches.

The remainder of this paper is organized as follows. We formally define SCs in Section 2, and study a number of fundamental problems (inference system, consistency, and implication) in Section 3. Section 4 presents how to detect SC violations. Section 5 discusses how to drill down into erroneous records. Section 6 reports our experimental findings. Related work is reviewed in Section 7 and we present conclusion and future work in Section 8.

2. STATISTICAL CONSTRAINTS

In this section, we provide some background definitions, and a formal definition of statistical constraints. We describe how CODED leverages SCs for error-detection.

2.1 Background Definitions

A **variable** X is an attribute or feature that can be assigned a value x from a fixed domain; we write $X = x$ to denote an assignment of a value to a variable. We use boldface vector notation for finite sets of objects. So for example $\boldsymbol{X} = \boldsymbol{x} \equiv (X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n)$ denotes the **joint assignment** where variable X_i is assigned value x_i, for each $i = 1, \ldots, n$. In relational terms, a variable corresponds to an attribute or column, and a joint assignments to a tuple or row.

A **random variable** requires a distribution $P(X = x)$ that assigns a probability to each domain value in the domain of X. A **joint distribution** $P(\boldsymbol{X} = \boldsymbol{x})$ assigns a

[1] In fact, this kind of error has happened in a Kaggle competition. A team found this error in the training set and utilized the dependence relationship between ID and class label to win the competition [2].

probability to each joint assignment. We often abbreviate the probability assignment $P(\boldsymbol{X} = \boldsymbol{x}) \equiv P(\boldsymbol{x})$ when the relevant set of variables is clear from context.

Given a joint distribution for a set of variables $\boldsymbol{X}$, the **marginal distribution** over a subset $\boldsymbol{Y}$ is defined by

$$P(\boldsymbol{Y} = \boldsymbol{y}) \equiv \sum_{\boldsymbol{z}} P(\boldsymbol{Y} = \boldsymbol{y}, \boldsymbol{Z} = \boldsymbol{z}).$$

Here $\boldsymbol{Z} = \boldsymbol{X} - \boldsymbol{Y}$ contains the set of variables outside of $\boldsymbol{Y}$, and the comma notation $\boldsymbol{Y} = \boldsymbol{y}, \boldsymbol{Z} = \boldsymbol{z}$ denotes the conjunction of two joint assignments. Here and elsewhere in this paper, the summation applies to variables with discrete domains, and should be replaced by integrals for variables with continuous domains. The **conditional probability** of an assignment $\boldsymbol{X} = \boldsymbol{x}$ given another assignment $P(\boldsymbol{Y}) = \boldsymbol{y}$ is defined as

$$P(\boldsymbol{X} = \boldsymbol{x} | \boldsymbol{Y} = \boldsymbol{y}) \equiv P(\boldsymbol{X} = \boldsymbol{x}, \boldsymbol{Y} = \boldsymbol{y}) / P(\boldsymbol{Y} = \boldsymbol{y}).$$

A key notion of this paper is the concept of *conditional independence among sets of variables*. Intuitively, a set of variables $\boldsymbol{X}$ is independent of another set $\boldsymbol{Y}$ given a third conditioning set $\boldsymbol{Z}$ if knowing the values of the variables in $\boldsymbol{Y}$ adds no information about the values of the variables in $\boldsymbol{X}$, beyond what can be inferred from the values in the set $\boldsymbol{Z}$. Formally, for three disjoint sets $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}$ we define

$$\begin{aligned} \boldsymbol{X} \perp\!\!\!\perp \boldsymbol{Y} | \boldsymbol{Z} \quad & \equiv \\ P(\boldsymbol{X} = x, \boldsymbol{Y} = y | \boldsymbol{Z} = z) \quad & = \\ P(\boldsymbol{X} = x | \boldsymbol{Z} = z) \times P(\boldsymbol{Y} = y | \boldsymbol{Z} = z) \quad & \text{for all } x, y, z \end{aligned}$$

This definition assumes that $P(\boldsymbol{Z} = \boldsymbol{z}) > 0$ for all possible values $\boldsymbol{z}$.

We call $\boldsymbol{X} \perp\!\!\!\perp \boldsymbol{Y} | \boldsymbol{Z}$ a **(conditional) independence statement**. The negation of a conditional independence statement is a **(conditional) dependence statement**, written $\boldsymbol{X} \not\perp\!\!\!\perp \boldsymbol{Y} | \boldsymbol{Z}$. Thus conditional dependence holds if for *some* values $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$, we have $P(\boldsymbol{Y} = \boldsymbol{y} | \boldsymbol{Z} = z) > 0$ and

$$P(\boldsymbol{X} = x | \boldsymbol{Y} = y, \boldsymbol{Z} = z) \neq P(\boldsymbol{X} = x | \boldsymbol{Z} = z).$$

2.2 Statistical Constraints

A set of Statistical Constraints (SCs) comprises a set of independence statements and dependence statements.

Definition 1 (Statistical Constraints). *Fix a set of variables $\boldsymbol{V} = \{V_1, \ldots, V_n\}$. A finite set of statistical constraints $\Sigma = \mathcal{I} \cup \mathcal{D}$ comprises*

1. *a finite set of independence* SC*s, $\mathcal{I} = \{\phi_1, \ldots, \phi_p\}$, where each ϕ_i is of the form $\boldsymbol{X} \perp\!\!\!\perp \boldsymbol{Y} | \boldsymbol{Z}$, and*
2. *a finite set of dependence* SC*s, $\mathcal{D} = \{\lambda_1, \ldots, \lambda_q\}$, where each λ_i is of the form $\boldsymbol{X} \not\perp\!\!\!\perp \boldsymbol{Y} | \boldsymbol{Z}$*

Please note in this paper, we investigate the statistical constraints among two or three variables. We will extend our approaches to more than three variables in the future work. The main idea of this paper is to detect data quality problems by finding contradictions with domain knowledge supplied by the user in the form of dependence and independence constraints. Such statistical constraints have several advantages.

Interpretability. (In)dependencies are easily interpreted by the user as ir(relevance) among attributes. Although the probabilistic semantics of a statistical constraint refers to a quantitative probability distribution, specifying an (in)dependence constraint does not require the user to specify or even consider numeric values. The large field of graphical models is based on the insight that (in)dependence constraints can be represented in terms of purely qualitative graphical relations among variables [18].

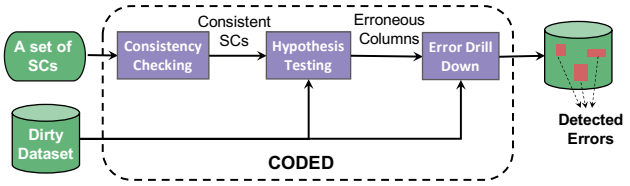

Figure 2: CODED Architecture

Detectability. The field of statistical hypothesis testing has developed many methods for deciding whether a given data set violates an independence constraint [42]. These methods provide parameters for controlling false positive and false negative error rates.

Expressive power. Deterministic constraints such as functional dependencies are powerful but also limited in the relationships among attributes that they allow a user to express. Often they reflect definitions and data recording conventions. In contrast, many important empirical facts about a domain can be expressed as statistical (in)dependencies among attributes. The next section considers the relationship between statistical and integrity constraints in some detail.

2.3 Integrity vs Statistical Constraints

Despite differences in tradition and terminology, integrity and statistical constraints often share underlying intuitions. Generally, integrity constraints enforce Boolean conditions on *sets* (relations), whereas statistical constraints impose (in)equalities on (conditional) *distributions*. For example, a functional constraint $\boldsymbol{X} \rightarrow Y$ entails an independence constraint $\boldsymbol{X} \perp\!\!\!\perp Y | \boldsymbol{X} - \{Y\}$. Whereas the functional constraint says that the variables $\boldsymbol{X}$ determine a unique *value* for Y regardless of the $\boldsymbol{Z}$ values, the independence constraint says that the variables $\boldsymbol{X}$ determine a unique *distribution* for Y regardless of the $\boldsymbol{Z}$ values. For another example, a marginal independence constraint $X \perp\!\!\!\perp Y$ entails a *multi-valued dependency* between columns X and Y [17]. Unlike integrity constraints, the traditional application of statistical constraints is to sample data, where we expect statistical constraints to hold only approximately with exceptions. As will be discussed in Section 4, this is accomplished by defining *degrees of dependence* (e.g. correlations in (0,1]).

2.4 CODED Architecture

Figure 2 shows the architecture of our SC-based error-detection system. The system consists of three key components: *consistency checking*, *hypothesis testing*, and *error drill down*. In the following, we present an overview of each component.

Consistency Checking. CODED takes a set of SCs and a dirty dataset as input. It first checks whether the given SCs are consistent or not. If not, it helps the user to resolve the conflict. For example, consider the four SCs in Figure 3. As will be explained in Section 3, they are actually inconsistent,

ID	Model	Color	Price	Fuel
r1	BMW X1	White	31000	25
r2	BMW X1	Black	32000	26
r3	BMW X1	White	33000	24
r4	BMW X1	Black	17000	22
r5	Toyota Prius	Black	18000	30
r6	Toyota Prius	White	19500	29
r7	Toyota Prius	White	18300	31
r8	Toyota Prius	White	18300	30

SC_1 = Model ⫫ Color SC_2 = Fuel ⫫ Model

SC_3 = Price ⫫ Fuel | Model SC_4 = Price ⫫̸ Fuel

Figure 3: A Running Example (SC_2 could be removed. The highlighted cells are the data errors detected by CODED w.r.t. SC_4.).

i.e., SC_2 and SC_3 implies Price ⫫ Fuel, which contradicts SC_4. CODED first tells the user that $\{SC_1, SC_2, SC_3, SC_4\}$ are inconsistent, and then guides the user to resolve the conflict, e.g., removing SC_2.

Hypothesis Testing. Once a set of consistent SCs are obtained, the system enters the hypothesis-testing stage. In this stage, the system checks for each SC, whether the dataset violates the SC or not. For example, consider SC_1 = Model ⫫ Color and the dataset in Figure 3. The user expects that Model and Color should have an independent relationship. However, a χ^2 hypothesis test on the data indicates that Model and Color are highly correlated. The system will mark Model and Color as erroneous columns. Section 4 below discusses hypothesis testing process.

Error Drill Down. A user may want to drill down into individual records so that she can understand why an SC is violated. Therefore CODED provides an error-drill-down component. The user chooses a violated SC and specifies a threshold k. This component aims to return k records that are most likely to cause the SC's violation. Suppose SC_2 = Price ⫫̸ Fuel is violated. As will be shown in Section 5, if the user specifies $k = 1$, the system will return r_4, which is a cheap car with high fuel consumption. The user can examine this record and may find out that unlike the other cars, this is a used car. It is worth noting that given SC_2 the user does not need to write a handcrafted rule like *a cheaper car tends to be more fuel-efficient*.

3. PROPERTIES OF SCs

Before applying SCs to error detection, we need to investigate a number of basic properties of SCs. In this section, we first present an inference system for SCs, and then study two fundamental properties associated with SCs: (1) *Consistency*: whether a set of SCs have conflicts; (2) *Implication*: whether a set of SCs imply (entail) another SC.

3.1 An Inference System for SCs

An inference system consists of a collection of inference rules, where each rule is in the form of

$$\mathrm{SC}_1 \ \& \ \mathrm{SC}_2 \cdots \ \& \ \mathrm{SC}_n \implies \mathrm{SC}_{new},$$

which states that if the left-hand side SCs hold, then the right-hand side SC must hold. Having an inference system is essential to solve the consistency problem. For example, suppose we are given three constraints:

$$\Sigma = \{\mathrm{SC}_1 : A \perp\!\!\!\perp B, \quad \mathrm{SC}_2 : A \perp\!\!\!\perp C|B, \quad \mathrm{SC}_3 : A \not\perp\!\!\!\perp C\}.$$

Applying the rule (see Definition 3):

$$(A \perp\!\!\!\perp B) \ \& \ (A \perp\!\!\!\perp C|B) \implies A \perp\!\!\!\perp C,$$

shows that Σ is inconsistent because SC_1 and SC_2 imply $\neg\mathrm{SC}_3$.

For existing row-oriented constraints, FDs have a sound and complete inference system that is known as Armstrongs Axioms [4]. Wenfei et al. extend Armstrongs Axioms to CFDs and prove that the new inference system is also sound and complete [8]. Xu et al. develop a set of sound (but not complete) inference rules for DCs [13]. A natural question is: can we develop a sound and complete inference system for SCs? In the following, we first introduce some background knowledge about the well-known *Graphoid Axioms* [18], and then present our inference system derived from the axioms.

Graphoid Axioms. Geiger and Pearl [18] showed that conditional independence statements satisfy the following axioms.

DEFINITION 2. *Let $\boldsymbol{X}$, $\boldsymbol{Y}$, $\boldsymbol{Z}$, and $\boldsymbol{W}$ denote four disjoint sets of random variables. The* graphoid Axioms $\mathcal{A}$ *are the following:*
Symmetry*:*

$$\boldsymbol{X} \perp\!\!\!\perp \boldsymbol{Y} \mid \boldsymbol{Z} \iff \boldsymbol{Y} \perp\!\!\!\perp \boldsymbol{X} \mid \boldsymbol{Z}$$

Decomposition*:*

$$\boldsymbol{X} \perp\!\!\!\perp \boldsymbol{YW} \mid \boldsymbol{Z} \implies \begin{cases} \boldsymbol{X} \perp\!\!\!\perp \boldsymbol{Y} \mid \boldsymbol{Z} \\ \boldsymbol{X} \perp\!\!\!\perp \boldsymbol{W} \mid \boldsymbol{Z} \end{cases}$$

Weak Union*:*

$$\boldsymbol{X} \perp\!\!\!\perp \boldsymbol{YW} \mid \boldsymbol{Z} \implies \boldsymbol{X} \perp\!\!\!\perp \boldsymbol{Y} \mid \boldsymbol{ZW}$$

Contraction*:*

$$(\boldsymbol{X} \perp\!\!\!\perp \boldsymbol{Y} \mid \boldsymbol{Z}) \ \& \ (\boldsymbol{X} \perp\!\!\!\perp \boldsymbol{W} \mid \boldsymbol{YZ}) \implies \boldsymbol{X} \perp\!\!\!\perp \boldsymbol{YW} \mid \boldsymbol{Z}$$

Note that $\boldsymbol{Z}$ can be the empty set. For example, for the symmetry axiom, we have $\boldsymbol{X} \perp\!\!\!\perp \boldsymbol{Y} \implies \boldsymbol{Y} \perp\!\!\!\perp \boldsymbol{X}$.

An Inference System for SCs. Graphoid Axioms $\mathcal{A}$ are applicable to any conditional independence statement. Since this paper only considers those conditional independence statements with no more than three variables, we can simplify $\mathcal{A}$.

DEFINITION 3. *Let A, B, C denote three random variables. The inference system $\mathcal{G}$ for* SCs *consists of the following rules.*
Symmetry*:*

$$A \perp\!\!\!\perp B \mid C \iff B \perp\!\!\!\perp A \mid C$$

Contraction-weak-union-decomposition*:*

$$(A \perp\!\!\!\perp B) \ \& \ (A \perp\!\!\!\perp C \mid B) \implies \begin{cases} A \perp\!\!\!\perp C \\ A \perp\!\!\!\perp B \mid C \end{cases}$$

A	T	T	T	T	F	F	F	F
B	T	F	T	F	T	F	T	F
C	T	T	F	F	T	T	F	F
P(A,B,C)	0.3	0.1	0.05	0.05	0.05	0.1	0.1	0.25

Figure 4: An example of a joint distribution that satisfies $\Sigma = \{A \perp\!\!\!\perp B,\ A \not\perp\!\!\!\perp B|C\}$

We can prove that Definition 2 and Definition 3 are equivalent for SCs involving at most three random variables. Therefore, we only need to consider the two inference rules in $\mathcal{G}$ rather than all the rules in $\mathcal{A}$. Lemma 1 formally proves their equivalence.

LEMMA 1. *For any set $\mathcal{I}$ of* SC*s involving at most three random variables, and any single* SC *γ, we have that (i) if $\mathcal{I} \vdash_{\mathcal{A}} \gamma$, then $\mathcal{I} \vdash_{\mathcal{G}} \gamma$; (2) if $\mathcal{I} \vdash_{\mathcal{G}} \gamma$, then $\mathcal{I} \vdash_{\mathcal{A}} \gamma$, where $\mathcal{I} \vdash_{\mathcal{A}} \gamma$ (resp. $\mathcal{I} \vdash_{\mathcal{G}} \gamma$) denotes that $\mathcal{I}$ implies γ using $\mathcal{A}$ (resp. $\mathcal{G}$).*

PROOF. All the proofs of this paper can be found in Appendix. □

We next discuss the soundness and completeness of the new inference system $\mathcal{G}$.

Soundness. Graphoid Axioms $\mathcal{A}$ have been proved to be sound [34]. Since $\mathcal{G}$ and $\mathcal{A}$ are equivalent for SCs (Lemma 1), we conclude that $\mathcal{G}$ is also sound.

Completeness. When Graphoid Axioms were first developed, they were conjectured to be a complete inference system [34]. In 1990s, Studeny proved that the conjecture [38] was *incorrect*. However, Studeny's proof assumes that each independence statement is allowed to contain an arbitrary number of random variables, thus it is still an open question whether the system $\mathcal{A}$ is complete under the assumption that each independence statement contains no more than three variables. For this reason, we conjecture that *when each independence statement contains no more than three variables, Graphoid Axioms $\mathcal{A}$ are complete.* If the conjecture is true, Lemma 1 entails that $\mathcal{G}$ is complete.

3.2 Consistency and Implication

Being able to check whether a given set Σ of SCs is consistent or not is important because if Σ is not consistent, a user will get contradictory error-detection results. In the following, we first formally define the consistency problem, and then present a consistency-checking framework. In order to apply the framework, the key is to solve the implication problem. Therefore, we define the implication problem and propose an efficient implication algorithm. Intuitively, if Σ is consistent, it means that Σ has no conflict. That is, there exists a joint distribution that satisfies Σ. Definition 4 presents a formal definition.

DEFINITION 4 (CONSISTENCY). *Given a set Σ of* SC*s, let $\boldsymbol{V}$ denote the set of random variables that appear in Σ. The consistency problem is to determine whether there exists a joint distribution $P(\boldsymbol{V})$ that satisfies every* SC *in Σ.*

For example, consider $\Sigma = \{A \perp\!\!\!\perp B,\ A \not\perp\!\!\!\perp B|C\}$. Since there are three random variables in Σ, we have $\boldsymbol{V} = \{A, B, C\}$. As shown in Figure 4, we can construct a joint distribution $P(A, B, C)$ that satisfies Σ, thus Σ is consistent. If we are given $\Sigma' = \{A \perp\!\!\!\perp B,\ A \not\perp\!\!\!\perp B\}$, it is impossible to construct a joint distribution that satisfies Σ', thus Σ' is inconsistent.

Consistency-checking Framework. Geiger and Pearl [18] proposed a framework to solve the consistency problem. They first divide Σ into two disjoint sets $\mathcal{I}$ and $\mathcal{D}$, where $\mathcal{I}$ (resp. $\mathcal{D}$) represents the set of independence (resp. dependence) statements in Σ. For each $\lambda \in \mathcal{D}$, it checks whether $\mathcal{I}$ implies $\neg\lambda$. If the answer is no for all λ, then Σ is consistent; otherwise, it is not consistent. The correctness of the framework was proved in [18]. For example, consider $\Sigma = \{A \perp\!\!\!\perp B,\ A \not\perp\!\!\!\perp B|C\}$. We have $\mathcal{I} = \{A \perp\!\!\!\perp B\}$ and $\mathcal{D} = \{A \not\perp\!\!\!\perp B|C\}$. We can see that $\mathcal{D}$ does not imply $\neg\lambda = A \perp\!\!\!\perp B|C$, thus Σ is consistent.

To apply the above framework, we aim to determine whether an independence SC γ can be inferred from $\mathcal{I}$ using $\mathcal{G}$. Definition 5 formally defines the implication problem.

DEFINITION 5 (IMPLICATION). *Given a set $\mathcal{I}$ of independence* SC*s, and another independence* SC *γ, the implication problem is to determine whether γ can be inferred from $\mathcal{I}$ using $\mathcal{G}$, i.e., check whether $\mathcal{I} \vdash_{\mathcal{G}} \gamma$ holds or not.*

Implication Algorithm. We propose an iterative algorithm. The algorithm starts with $\mathcal{I}^* = \mathcal{I}$, and then iteratively expands $\mathcal{I}^*$ with new SCs. At each iteration, the algorithm checks which inference rule in $\mathcal{G}$ can be used to generate a new SC, and then adds it to $\mathcal{I}^*$. The algorithm terminates when $\mathcal{I}^*$ cannot be further expanded. After that, the algorithm checks whether $\gamma \in \mathcal{I}^*$ holds. If so, then $\mathcal{I}$ implies γ; otherwise, $\mathcal{I}$ does not imply γ.

EXAMPLE 1. *Given a set of independence* SC*s $\mathcal{I} = \{A \perp\!\!\!\perp B, A \perp\!\!\!\perp C|B, B \perp\!\!\!\perp C, C \perp\!\!\!\perp E|D\}$ over attributes $M = \{A, B, C, D, E\}$. To compute $\mathcal{I}^*$, we initially set $\mathcal{I}^* = \mathcal{I}$, and then apply the implication algorithm.*
Step 1: Constructing Hash Table. $H = \{\{A, B\}, \{B, C\}\}$ will be generated according to the marginal SC*s in $\mathcal{I}^*$, i.e., $A \perp\!\!\!\perp B$ and $B \perp\!\!\!\perp C$.*
Step 2: Employing Inference System $\mathcal{G}$. We check each conditional SC *in $\mathcal{I}^*$. For $\text{SC}_1 : A \perp\!\!\!\perp C|B$, we first check whether $\{A, B\}$ or $\{B, C\}$ exists. Obviously, we have both of them in the H. Applying the* contraction-weak-union-decomposition *inference rule in $\mathcal{G}$, we can obtain three new* SC*s: $A \perp\!\!\!\perp B|C$, $B \perp\!\!\!\perp C|A$, and $A \perp\!\!\!\perp C$. We add them into $\mathcal{I}^*$, and hash $A \perp\!\!\!\perp C$ as $\{A, C\}$ and add it into H.*
Step 3: Repeating. Repeat Step 1 and Step 2 till there are no more SC*s that can be inferred. In this example, we eventually arrive at $\mathcal{I}^* = \{A \perp\!\!\!\perp B, A \perp\!\!\!\perp C, B \perp\!\!\!\perp C, A \perp\!\!\!\perp B|C, A \perp\!\!\!\perp C|B, B \perp\!\!\!\perp C|A, C \perp\!\!\!\perp E|D\}$.*

To analyze the time complexity of the implication algorithm, we answer two questions:
Q1. What's the time complexity of generating a new SC*?*
Q2. How many new SC*s can be generated in total?*
For $Q1$, consider the two inference rules in Definition 3. The algorithm first checks whether the symmetry rule can be used to generate a new SC or not. The time complexity of this process is $\mathcal{O}(|\mathcal{I}^*|)$. For the contraction-weak-union-decomposition rule, a naive way is to enumerate every pair of SCs in $\mathcal{I}^*$ and then check whether the rule can be used or not. It needs $\mathcal{O}(|\mathcal{I}^*|^2)$ time. In fact, we can utilize a hash table to reduce the time complexity to $\mathcal{O}(|\mathcal{I}^*|)$. Specifically, we build a hash table for marginal SCs (e.g., $A \perp\!\!\!\perp B$),

Algorithm 1: Consistency-checking Algorithm

Input: A set of SCs: Σ
Output: Consistent or not (YES or NO)

```
1 Divide Σ into I and D;
2 Use the implication algorithm to obtain I*;
3 for λ ∈ D do
4 |   if ¬λ ∈ I* then
5 |   |   return NO;
6 return YES;
```

and then for each conditional SC (e.g., $A \perp\!\!\!\perp C|B$), we check whether $A \perp\!\!\!\perp B$ or $C \perp\!\!\!\perp B$ exists in the hash table.

For $Q2$, we seek to compute an upper bound for the number of newly generated SCs. The key observation is that for any inference rule in $\mathcal{G}$, its right-hand side will not introduce any new variable that is not on the left-hand side. For example, consider the contraction-weak-union-decomposition rule. The left-hand side has three variables $\{A, B, C\}$, and the right-hand side has the same three variables, without introducing any new variable.

Based on this observation, we find that there is a relationship between $|\mathcal{I}|$ and $|\mathcal{I}^*|$. That is, $|\mathcal{I}^*| \leq 12|\mathcal{I}|$. If we consider symmetry SCs (e.g., $A \perp\!\!\!\perp B$ and $B \perp\!\!\!\perp A$) are the same, then $|\mathcal{I}^*| \leq 6|\mathcal{I}|$. We give the intuition below, and provide a formal proof in Lemma 2.

We say SC_1 *covers* SC_2 if the variable set of SC_1 is a superset of the variable set of SC_2. For example, suppose $\text{SC}_1 : A \perp\!\!\!\perp B|C$. Then, it covers 6 different SCs:

$$A \perp\!\!\!\perp B,\ \ A \perp\!\!\!\perp C,\ \ B \perp\!\!\!\perp C,\ \ A \perp\!\!\!\perp B|C,\ \ A \perp\!\!\!\perp C|B,\ \ B \perp\!\!\!\perp C|A$$

The above observation implies the SCs in $\mathcal{I}$ can cover all the SCs in $\mathcal{I}^*$. Since each $\text{SC} \in \mathcal{I}$ can cover at most 6 different SCs, then we have that $|\mathcal{I}^*| \leq 6|\mathcal{I}|$. Lemma 2 formally proves this result.

LEMMA 2. *Given a set $\mathcal{I}$ of independence* SC*s, let $\mathcal{I}^*$ denote the set of all the independence* SC*s implied from $\mathcal{I}$. Then, we have $|\mathcal{I}^*| \leq 6|\mathcal{I}|$.*

Consistency-checking Algorithm. We can develop a consistency-checking algorithm using the implication algorithm. See the pseudo-code in Algorithm 1. The algorithm first computes $\mathcal{I}^*$. Then, for each $\lambda \in \mathcal{D}$, it checks whether $\neg\lambda \in \mathcal{I}^*$ holds or not. If no for all λ, then Σ is consistent; otherwise, Σ is inconsistent.

EXAMPLE 2. *Given a set of* SC*s $\Sigma = \{A \perp\!\!\!\perp B, A \perp\!\!\!\perp C|B, A \not\perp\!\!\!\perp C, B \perp\!\!\!\perp C, C \not\perp\!\!\!\perp D, C \perp\!\!\!\perp E|D, E \not\perp\!\!\!\perp F\}$, we check the consistency of Σ as follows:*
Step 1: Dividing. Divide Σ into $\mathcal{I} = \{A \perp\!\!\!\perp B, A \perp\!\!\!\perp C|B, B \perp\!\!\!\perp C, C \perp\!\!\!\perp E|D\}$ and $\mathcal{D} = \{A \not\perp\!\!\!\perp C, C \not\perp\!\!\!\perp D, E \not\perp\!\!\!\perp F\}$;
Step 2: Computing $\mathcal{I}^*$. By employing the implication algorithm on $\mathcal{I}$, we are able to acquire $\mathcal{I}^*$. Similar to Example. 1, we can obtain $\mathcal{I}^* = \{A \perp\!\!\!\perp B, A \perp\!\!\!\perp C, B \perp\!\!\!\perp C, A \perp\!\!\!\perp B|C, A \perp\!\!\!\perp C|B, B \perp\!\!\!\perp C|A, C \perp\!\!\!\perp E|D\}$.
Step 3: Checking the Implication. Check that whether there exists $\lambda \in \mathcal{D}$ such that $\neg\lambda \in \mathcal{I}^*$ holds. We see that $\neg\lambda : A \perp\!\!\!\perp C$ is in $\mathcal{I}^*$, thus Σ is inconsistent.

Time Complexity. The implication algorithm needs $\mathcal{O}(|\mathcal{I}^*|)$ iterations, and each iteration needs $\mathcal{O}(|\mathcal{I}^*|)$ time. The total time complexity is $\mathcal{O}(|\mathcal{I}^*|^2) = \mathcal{O}((6|\mathcal{I}|)^2) = \mathcal{O}(|\mathcal{I}|^2)$. For the consistency-checking algorithm, the main bottleneck is spent in running the implication algorithm to compute $\mathcal{I}^*$ (see Line 2 in Algorithm 1), thus the total time complexity is $\mathcal{O}(|\mathcal{I}|^2)$.

Resolving Inconsistency. Once an inconsistency is detected, we help the user to resolve the inconsistency. Specifically, suppose the consistency-checking algorithm finds a conflict: $\mathcal{I}$ implies $A \perp\!\!\!\perp B$ but $A \not\perp\!\!\!\perp B$ is in Σ. We ask the user to check which one is correct. If the former is correct, then $A \not\perp\!\!\!\perp B$ should not be in Σ, and we remove $A \not\perp\!\!\!\perp B$ from Σ. Otherwise, we show the derivation of $A \perp\!\!\!\perp B$ from $\mathcal{I}$ and then ask the user to remove the incorrect SCs involved in the derivation.

4. DETECTING SC VIOLATION

Evaluating whether a dataset satisfies an independence constraint is a classic application of statistical hypothesis testing. Hypothesis testing has been researched by statisticians for over a century, and is well covered in a number of sources [42, Ch.10]. To make the paper self-contained, we will briefly review the basic ideas. Note that in traditional hypothesis testing, the assumptions are that data is clean and a hypothesis might be wrong. Thus, their use case is *not* to apply hypothesis testing to detect data errors. Our contribution is to bring the well-studied statistical hypothesis testing to this new (error detection) problem domain.

A **hypothesis test** T is a procedure that takes as input a dataset D and outputs either 0 ("the hypothesis is rejected") or 1 ("the hypothesis is not rejected"). In classic scientific methodology, a deterministic hypothesis is rejected if it is inconsistent with the data. A statistical hypothesis is rejected when the probability of the data entailed by the hypothesis is below a user-specified threshold. Most hypothesis tests are based on test statistics. A **test statistic** $\phi(D)$ returns a real number for a given data set. Intuitively, the statistic is an aggregate function that summarizes the degree to which the dataset violates the independence hypothesis $\boldsymbol{X} \perp\!\!\!\perp \boldsymbol{Y}|\boldsymbol{Z}$. The **$p$-value** specifies the probability of observing a value at least as great as the test statistic for the dataset, assuming the independence constraint:

$$p(D) = P(t \geq \phi(D)|\boldsymbol{X} \perp\!\!\!\perp \boldsymbol{Y}|\boldsymbol{Z}).$$

In most applications, the user specifies a threshold α (e.g. 5%), known as the **significance level**, and rejects the hypothesis if $p(D) < \alpha$. This procedure ensures that the probability of a false positive (rejection) is below α.

Which test statistics are suitable for independence hypotheses depends on the data type. In the following, we present commonly used statistics for the data types we consider in this paper. Due to the space constraint, we describe independence tests only for unconditional independence $\boldsymbol{X} \perp\!\!\!\perp \boldsymbol{Y}$. In the case where the conditioning variable set $\boldsymbol{Z}$ is discrete, the unconditional test is applied for each setting $\boldsymbol{Z} = \boldsymbol{z}$.

4.1 Categorical/Discrete Data

Given an assignment $\boldsymbol{X} = \boldsymbol{x}$, each datapoint satisfies the assignment or not. The **observed count** is the number $N_D(\boldsymbol{X} = \boldsymbol{x})$ of datapoints that satisfy it. If the sets $\boldsymbol{X}$ and $\boldsymbol{Y}$ were completely independent, the observed counts would equal the product of the marginal counts; these products

are called the **expected counts**: $E_D(\boldsymbol{X} = \boldsymbol{x}, \boldsymbol{Y} = \boldsymbol{y}) \equiv N_D(\boldsymbol{X} = \boldsymbol{x}) \times N_D(\boldsymbol{Y} = \boldsymbol{y})$. Test statistics for independence among discrete variables measure the difference between expected and observed counts. The most widely used statistic for categorical data is Pearson's **chi-square statistic** Q introduced in 1900. It sums the squared differences, weighted inversely to the expected counts:

$$Q(D) \equiv \sum_{\boldsymbol{x},\boldsymbol{y}} \frac{[N_D(\boldsymbol{X} = \boldsymbol{x}, \boldsymbol{Y} = \boldsymbol{y}) - E_D(\boldsymbol{X} = \boldsymbol{x}, \boldsymbol{Y} = \boldsymbol{y})]^2}{E_D(\boldsymbol{X} = \boldsymbol{x}, \boldsymbol{Y} = \boldsymbol{y})}.$$

The distribution of the chi-square statistic can be approximated by the χ^2 distribution. That is, we have $p(D) \approx \int_{Q(D)}^{+\infty} \chi^2_k(t)dt$ where $k = (r_X - 1) \times (r_Y - 1)$ and r_X resp. r_Y is the number of possible assignments for $\boldsymbol{X}$ resp. $\boldsymbol{Y}$. We use the chi-square statistic for testing independence among discrete (categorical) variables.

4.2 Numerical Data

We discuss independence tests for two numerical variables X and Y. The main idea is to consider the degree to which X and Y vary together: whether an increase in X tends to correspond to an increase or decrease in Y. A number of different formulas have been developed for capturing this idea. Perhaps the best known is Pearson's correlation coefficient ρ. The correlation coefficient measures the degree to which X and Y are linearly related. The disadvantage of the ρ statistic is that it is reliable only under certain assumptions, including that the relationship between X and Y is approximately linear. Since in the error detection problem, we have very little a priori knowledge of the exact form of the statistical dependence between X and Y, we examine statistical tests that make minimal assumptions about the nature of the dependence. Such tests are called *non-parametric*, since they do not assume that the dependence can be characterized by a fixed set of parameters known a priori before data observation.

A well-established set of non-parametric tests for dependence are known as **rank correlations**. Rank correlations are based on a straightforward intuition: Each variable defines an ordering over data points, and if the variables are associated, the X-ordering should carry information about the Y-ordering. In our car running example, if the most expensive car is also the most fuel-efficient, the second-most expensive car is the second-most fuel-efficient, ..., and the cheapest car is the least fuel-efficient, then there is a positive relationship between price and fuel efficiency. A rank correlation test, therefore, considers how similar the X-ranking is to the Y-ranking. A number of similarity metrics for rankings have been proposed; the most common non-parametric metric is Kendall's τ.

The definition of the τ statistic is as follows. Consider n datapoints with two features $D = (x_1, y_1) \ldots, (x_n, y_n)$. For now we assume there are no ties, meaning that all datapoints have different values for X and different values for Y. Then for two datapoints (x_i, y_i) and (x_j, y_j) with $i \neq j$, there are two possibilities.

1. $x_i > x_j$ and $y_i > y_j$, or $x_i < x_j$ and $y_i < y_j$. In this case the two variables agree on the ordering of i and j and the pair (i, j) is **concordant**. The number of concordant pairs is denoted as $n_c(D)$.

Table 2: Supported data types and default hypothesis testing methods

A	B	C	Default Test
Categorical	Categorical	×	χ^2 Test
Categorical	Categorical	Categorical	χ^2 Test
Numerical	Numerical	×	τ Test
Numerical	Numerical	Categorical	τ Test

2. $x_i > x_j$ and $y_i < y_j$, or $x_i < x_j$ and $y_i > y_j$. In this case the two variables disagree on the ordering of i and j and the pair (i, j) is **discordant**. The number of discordant pairs is denoted as $n_d(D)$.

The τ statistic is then computed as

$$\tau(D) \equiv \frac{n_c(D) - n_d(D)}{\binom{n}{2}}.$$

The intuition behind the τ statistic is that it represents the difference in the proportion of concordant pairs and the proportion of discordant pairs. To compute p-values, it is possible to define a rescaled denominator, still a function of only n, such that the rescaled statistic has an approximately Gaussian distribution.

If a dataset contains tied datapoints, a modified τ_b statistic can be used. Essentially, τ_b computes the difference in the number of concordant and discordant pairs without ties, and rescales the denominator to achieve an approximately Gaussian distribution. In our experiments below, we use τ_b for datasets with ties, but still refer to τ for simplicity. Table 2 summarizes the hypothesis testing method and the data types supported by them.

5. ERROR DRILL DOWN

In this section, we study how to drill down into individual records that are most likely to cause the violation of an SC. We first propose a general framework in Section 5.1 and then propose two efficient top-k algorithms for categorical data and numerical data, respectively, in Section 5.2

5.1 Error-Drill-Down Framework

There are two reasons that motivate the drill-down task. Firstly, an SC violation helps a user to detect which columns have errors, but it does not tell the user which values in the columns may contain the errors. For example, suppose a user specifies an SC: Model $\perp\!\!\!\perp$ Color on a car dataset. After a hypothesis test, she finds that Model and Color violate the independent relationship on the dataset. At this point, she knows that there may be some errors in the Model and Color columns. We want to help her to locate the errors in the columns so that she can figure out why the violation happens and then fix the errors.

Secondly, sometimes a dataset has errors, but the errors are not of sufficient magnitude to violate a (in)dependence relationship among columns. We want to leverage SCs to detect the errors even in this situation. For example, consider an SC: Fuel $\not\perp\!\!\!\perp$ Price. Suppose that after a hypothesis test, the test result indicates that that Fuel and Price are dependent (i.e., no violation) but the dependent relationship is weaker than what she expects. We want to help her investigate why the correlation is weaker than expected.

To this end, we develop an interactive error-drill-down framework. A user specifies an SC and a hypothesis testing method (e.g., τ test). The framework first applies the

hypothesis test to the data, and checks whether SC is violated. If yes, it will return k records whose column values are most likely to cause the violation. If no, a user can check whether the returned statistic (e.g., the τ statistic) is larger or smaller than her expectation. Suppose the τ statistic is smaller than what she expects. Then, she can use the framework to identify k records whose column values are most likely to cause the unexpected result.

Central to this framework is the *top-k error detection problem*. Let D denote a dataset, and S denote a hypothesis testing statistic (e.g., the τ statistic or the χ^2 statistic). Without loss of generality, we consider only the case where the statistic is larger than a user's expectation. In this situation, the goal is to find k records from D such that if they were removed, the test statistic would decrease the most. Definition 6 formally defines the problem.

DEFINITION 6 (TOP-k INDIVIDUAL ERROR DETECTION). *Given a dataset D, an* SC*, a hypothesis testing statistic S, and a threshold k, we aim to identify a set of k records, denoted by ΔD, such that $S(D-\Delta D)$ is mimized, i.e.,*

$$\underset{\Delta D \subseteq D}{\operatorname{argmin}} \; S(D-\Delta D) \quad \textit{s.t.} \quad |\Delta D| = k$$

A naive solution is to enumerate all $\binom{|D|}{k}$ possibilities, and then return the best result. This is prohibitively expensive. Even with a modest data size of 10,000 records, assuming a reasonable k = 10, this approach would require enumerating $\binom{10000}{10} = 2.7\times10^{33}$ possibilities. Therefore, we adopt greedy algorithms to reduce the cost. We propose two greedy algorithms, K *strategy* and K^c *strategy*. Intuitively, the K strategy seeks to directly identify the best k records; the K^c strategy seeks to remove the worst $n-k$ records and then return the remaining k records as a result.

K Strategy. The algorithm first selects the *best* record d^* from D such that if it was removed, the statistic can decrease the most. The algorithm removes d^* from D, and then repeats the above process to select the best record from $D-\{d^*\}$. After k iterations, the top-k records are identified.

K^c Strategy. The algorithm first selects the *worst* record d' from D such that if it was removed, the statistic can *increase* the most. The algorithm removes d' from D, and then repeats the above process to select the worse record from $D-\{d'\}$. After $n-k$ iterations, where $n=|D|$, the remaining k records are returned.

Remark. The K strategy is more efficient than the K^c strategy because the former only needs to select k records but the latter needs to check $n-k$ records. In terms of effectiveness, the K strategy often leads to a better objective value (i.e., smaller $S(D-\Delta D)$ because it directly optimizes for that value. The K^c strategy is particularly useful in identifying a set of k records that are highly correlated with each other, thus it is more suitable to detect errors for the violation of an independence SC.

5.2 Top-k Error Detection Algorithms

We describe top-k error detection methods for the two statistics we examine in this paper, χ^2 (Kendall's) τ. We discuss how to implement the K strategy for them efficiently. The same implementation can be extended to the K^c strategy trivially.

5.2.1 Categorical Data

	BMW X1	Toyota Prius
White	192	200
Black	213	189
Blue	54	369

Figure 5: Group counts of Model and Color

We use χ^2 test as default method to detect errors for categorical data. We propose an optimization technique to reduce the time of selecting the "best" record at each iteration. The key observation is that if two records have the same values on the tested columns, there is no difference to choose either one of them. Therefore, we can group the records based on the tested columns, and only need to spend time in deciding which group (rather than which record) should be selected at each iteration.

For example, consider an SC: Model $\perp\!\!\!\perp$ Color and a dataset similar to Figure 3 but with more records. We first group the records based on the tested columns (i.e., Model and Color). Suppose there are 2 car models and 3 colors. Then, there will be 6 groups in total. Figure 5 illustrates the 6 groups, where the number in each cell represents the total number of the records that belong to that group. Each cell has a χ^2 value z. At each iteration, the K strategy only needs to determine which group should be selected and then randomly picks one record from that group based on the statistic z. This optimization technique significantly reduces the computational cost since the number of groups could be orders of magnitude smaller than the total number of records.

5.2.2 Numerical Data

We discuss how we employ the framework for numerical data with the τ test. We formally define the problem as follows.

DEFINITION 7 (τ-TEST-BASED ERROR DETECTION). *Given a dataset D, an* SC*, and k, the τ-test-based error detection problem tries to find a subset ΔD of records from D that contribute the most to the violation of the* SC*, i.e.,*

$$\underset{\Delta D \subseteq D}{\operatorname{argmin}} \; \frac{n_c(D-\Delta D) - n_d(D-\Delta D)}{\binom{|D|-k}{2}} \quad \textit{s.t.} \quad |\Delta D| = k$$

For simplicity, we denote the count of concordant pairs, discordant pairs, and tied pairs in $D-\Delta D$ as n_c, n_d, and n_t, respectively. We omit the denominator of the objective function since it is a constant function of n only. The K strategy works as follows. At each iteration, it calculates the *benefit* of each record: Given a record r, find all the pairs that contain the record, and then calculate the sum of the weights of these pairs, denoted by benefit(r). At each iteration, we select the record with the biggest benefit. Once a record is selected, we need to update the benefit of all the remaining records. We use a priority queue to maintain the top-k records. Algorithm 2 shows the pseudo-code. Example 3 illustrates the algorithm.

EXAMPLE 3. *Following the running example in Figure 3, we apply the tau testing method to detect data errors w.r.t.* SC_1 = Price $\perp\!\!\!\perp$ Fuel*, and let $k=1$.*

Algorithm 2: τ-test-based error detection algorithm

Input: An SC, Dataset D, k
Output: k records

```
Q ← ∅; // Priority Queue
R ← ∅; // Returned List
W ← ∅; // A hash table that maps a pair to its weight
benefit(r) = 0 for r ∈ D ;
// Initialization
for r_i ∈ D do
    for r_{i+1} ∈ D do
        if r_i, r_j are concordant then
            benefit(r_i) += 2 ;
            benefit(r_{i+1}) += 2 ;
            W(r_i, r_{i+1}) = 2 ;
        else if r_i, r_j are tied then
            benefit(r_i) += 1 ;
            benefit(r_{i+1}) += 1 ;
            W(r_i, r_{i+1}) = 1 ;
        Q.push(⟨r_i, benefit(r_i)⟩);
// Iteration
for i = 1 to k do
    Add Q.top() to R;
    Update Q; // Update the benefit of each record r ∈ Q
      that has a non-zero weight with r_i (W(r, r_i) = 2 or 1)
return R;
```

Step 1: Initialization. We calculate the benefit of each record on the entire dataset, and obtain the priority queue $Q = \{r_4 : 0, r_5 : 9, r_7 : 9, r_1 : 10, r_2 : 10, r_8 : 10, r_3 : 12\}$. *Intuitively, the benefit of a record (e.g.,* benefit$(r_5) = 9$*) means that if* r_5 *was selected and removed, the objective value would be decreased by 9.*

Step 2: Removing and Updating. Return the top element of Q*, in this case* r_4*. Then remove* r_4*, and update the priority queue. Since* r_4 *shares a zero weight with every other record, the benefit of each record keeps unchanged. The updated queue will be* $Q = \{r_5 : 9, r_7 : 9, r_1 : 10, r_2 : 10, r_8 : 10, r_3 : 12\}$.

Step 3: Repeating Step 2. We repeat Step 2 until k *records are returned. Here, since* $k = 1$*, only* r_4 *will be returned.*

Efficiency Analysis. The main computational bottleneck of the K strategy is the initialization phase. Consider a dataset with two columns: $D = \langle x_1, y_1\rangle, \cdots, \langle x_n, y_n\rangle$. We need to initialize the benefit of each record. For a single record, this requires comparing with the other $n-1$ records, leading to a time complexity of $\mathcal{O}(n)$. Therefore the naive implementation of the initialization phase needs $\mathcal{O}(n^2)$ time. This does not scale to large data sets (e.g. 1M records).

We find that the time complexity of the initialization phase can be reduced to $\mathcal{O}(n \log n)$ with the help of a segment tree. A segment tree is a tree data structure, where each node stores information about a segment. It allows for inserting a segment and querying a segment with both $\mathcal{O}(\log n)$ time. To apply this idea, we first sort D by column X and then scan the records in D based on this new order. For each record $\langle x_i, y_i\rangle$, we can get the number of concordant pairs of the record by querying the segment of $(-\infty, y_i)$, and the number of discordant pairs by querying the segment of $(y_i, +\infty)$. Once the two numbers are obtained, we insert a segment $[y_i, y_i]$ (representing a single point) to the tree. We need $\mathcal{O}(\log n)$ time to process each record, thus the total time complexity is $\mathcal{O}(n \log n)$. With this optimization, error drill-down can handle a dataset with millions of records. Further details are in Appendix.

6. EXPERIMENTS

We evaluate the effectiveness and efficiency of our methods on real-life datasets with both synthetic errors and real errors. Specifically, we examine (1) the computational efficiency of the consistency-checking methods, (2) the effectiveness of our method compared to the state-of-the-art approaches on detecting synthetic and real-life errors, and (3) the scalability of our error-detection method.

6.1 Experiment Setup

Datasets. We used four real-life datasets to evaluate our approaches.

(1) `BOSTON`[2]. The Boston dataset was taken from the Boston Standard Metropolitan Statistical Area (SMSA) in 1970. This dataset was first used in [19] to study the relationship between clean air quality and household's willing to pay. There are 506 instances, and each instance has 14 attributes. We used 6 attributes: Distance to CBD area-Distance (D), Nitric Oxides Concentration-N_oxide (N), and Crime Rate-Crime (C), Black index of population(B), Rooms(R) and Tax Rate(T).

(2) `CAR`[3]. The Car Evaluation dataset is from UCI Machine Learning repository. This dataset contains seven attributes. We used 4 attributes: Buying price(BP), Car Class (CL), Doors(DR) and Safety level(SA).

(3) `SENSOR`[4]. The Sensor dataset collected the sensor reports from the Berkeley/Intel Lab. The dataset has more than 2 million records, containing the humidity and temperature reports from 54 different sensors.

(4) `HOCKEY`[5]. The Hockey dataset collected the records of each NHL game from 1998-2010. It has more than ten attributes which variously describe player attributes and player performance statistics for a season.

Simulated Errors. In Section 1, we introduced two types of errors: *sorting error* and *imputation error* (see Figure 1). We simulated them on the Boston dataset. We also simulate the *Imputation Error* on the CAR dataset.

To simulate the *sorting* error on column $A = \{a_1, a_2, \cdots, a_n\}$, we first select a sub-sequence of A, then sort them in an ascending order, denoted by $A' = \{a'_i, \cdots, a'_j\}$ ($a'_i \le a'_{i+1} \le \cdots \le a'_j$). After that, we put the sorted values back to the original column, i.e., $\overline{A} = \{a_1, \cdots, a_{i-1}, a'_i, \cdots, a'_j, a_{j+1}, \cdots, a_n\}$. We say that $\overline{A}$ has sorting error with the error rate of $\frac{j-i+1}{n}$. Note that the sorting error may either make two columns A and B more independent or less independent. If the original data is randomly shuffled, then $\overline{A}$ and B tend to be more independent; if the original data is sorted by B, then $\overline{A}$ and B tend to be less independent. We used the former for dependence SCs, the latter for independence SCs.

To simulate the imputation error on column $A = \{a_1, a_2, \cdots, a_n\}$, we assume that A has a set of m missing values (i.e., error rate $= \frac{m}{n}$), and the missing values are replaced by the mean

[2] https://www.cs.toronto.edu/ delve/data/boston/bostonDetail.html
[3] https://archive.ics.uci.edu/ml/datasets/Car+Evaluation.
[4] http://db.csail.mit.edu/labdata/labdata.html
[5] https://github.com/liuyejia/Model_Trees_Full_Dataset

Table 3: Constraints used by different error detection methods

Attributes	CODED	Denial Constraints
Rooms(R), Black Index(B)	$R \perp\!\!\!\perp B$	×
N_oxide(N), Distance(D)	$N \not\perp\!\!\!\perp D$	For any two records r_1 and r_2, if $r_1[N] > r_2[N]$, then $r_1[D] < r_2[D]$
N_oxide, Black Index, Tax rate (T)	$N \perp\!\!\!\perp B \mid T$	×
Tax rate, Black Index, Crime(C)	$T \not\perp\!\!\!\perp B \mid C$	For any r_1 and r_2 with $r_1[C] = r_2[C]$, if $r_1[T] > r_2[T]$, then $r_1[B] < r_2[B]$
Buying Price(BP), Class(Cl)	$BP \not\perp\!\!\!\perp Cl$	×
Safety(SA), Doors(DR)	$SA \perp\!\!\!\perp DR$	×
Temperatures (T) of Sensor 8 and Sensor 9	$T_8 \not\perp\!\!\!\perp T_9$	For any r_1 and r_2, if $r_1[T_8] > r_2[T_8]$, then $r_1[T_9] > r_2[T_9]$
Games(G), Goal Plus-Minus(GPM), Year(Y)	$G \perp\!\!\!\perp GPM \mid Y$	×

value of column A. Let $\overline{A}$ denote the new column. Like the sorting error, the imputation error may either make two columns more independent or less independent. If the missing values are randomly selected, $\overline{A}$ and B tend to become more independent; if the missing values are selected based on a range condition on B, $\overline{A}$ and B tend to become less independent. We used the former (latter) for dependence (independence) SCs.

We also explored the combined impact of the two error types because combining errors is not rare in practice. Our *combination error* consists of 80% sorting error and 20% imputation error.

Real-life Errors. For the Sensor dataset, we first aggregated the mean value of sensor reports by hours. Reported mean temperatures higher than 60 degrees or lower than 15 degrees were assumed to be erroneous and removed, as in [32]. We replaced these missing values by mean value imputation to obtain an input dataset. We adopted the method in [26] for defining outliers: Keep a time window of neighboring sensors, and label as an outlier any sensor report that deviates by more than a factor of 3 from the region's mean value. After the pre-processing, the goal becomes to to detect the imputed values and the outliers as erroneous.

For the Hockey dataset, the dataset creators performed imputations to support machine learning analysis. The imputations were found by comparing their dataset with the original data from hockey data websites. We aim to detect the imputed values.

Error-Detection Approaches. We compared CODED with two state-of-the-art error detection approaches, Denial Constraints(DC) [12] and DBoost [31]. Table 3 summarized the constraints used by CODED and DC. For the CODED hypothesis testing, we used the χ^2 test for categorical data, and the τ test for numerical data. We implemented the error-drill-down framework, and adopted the K strategy for dependence SCs and the K^c strategy for independence SCs. DC is a constraint-based error detection approach. We counted the DC violations of each record, and returned the top-k records that involve the most number of violations. DBoost is an outlier detection approach. We used an implementation available on-line[6]. We applied DBoost with three models: GMM, Gaussian and Histogram. For categorical data, we employed the bin width that achieves the best f-score results. For numeric data, we employed Gaussian and GMM with the mixture parameter n_subpops threshold set at $3, 0.001$, and the statistical epsilon to be 0.

Quality Measurement. We used *Precision@K*, *Precision*, and *Recall* to measure the quality of error-detection approaches. *Precision* is the ratio of the number of correctly detected records to the number of total returned records, *Recall* is the ratio of the number of correctly detected records to the number of total erroneous records. *Precision@K* is the precision among the k returned records.

[6] https://github.com/cpitclaudel/dBoost

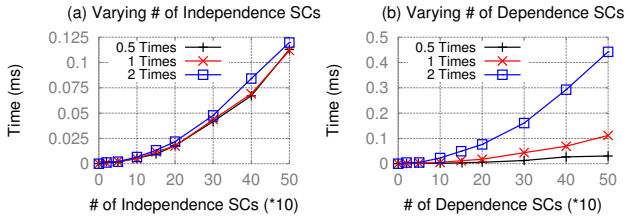

Figure 6: Time for consistency checking of SCs

6.2 Experimental Results

Exp-1: Efficiency of Consistency Checking. We examined the efficiency of the consistency-checking algorithm. Note that consistency checking is *not* related to data. Therefore, randomly generating SCs is appropriate to test the scalability of our consistency-checking algorithm. We randomly generated $|\mathcal{I}|$ independence SCs and $|\mathcal{D}|$ dependence SCs from $|\mathcal{D}|/2$ variables. We measured the runtime of consistency checking by varying $|\mathcal{I}|$ and $|\mathcal{D}|$, respectively. The results are shown in Figure 6. Each figure has three lines, which represent different ratios of $|\mathcal{I}|$ to $|\mathcal{D}|$.

We make two interesting observations. First, our consistency-checking algorithm scales well by varying either $|\mathcal{I}|$ or $|\mathcal{D}|$. Even with 500 independence (dependence) SCs, the algorithm can terminate within 0.5 ms. Second, the efficiency is more related to $|\mathcal{I}|$ than $|\mathcal{D}|$. This is because the consistency-checking algorithm proposed comprises two parts: implication and checking, where the implication algorithm needs $\mathcal{O}(|\mathcal{I}^2|)$ time, which typically dominates the whole process.

Exp-2: Impact of Errors. To illustrate the impacts of data errors on independence and dependence relationships among variables, we varied the rates of sorting error, imputation error, and combination error, and tested how the τ value changed accordingly.

Figure 7 shows the results w.r.t. an independence SC. All errors tend to raise the tau value, indicating a violation of the independence SC. The sorting error has a consistent effect on the tau value (Figure 7(a)). Imputation error has a strong negative impact at lower error rates, then the tau value reverses (Figure 7(b)). This can be explained as follows. Starting with independent data, imputing a constant value induces correlations among the two columns. Once many data points have been imputed with a constant value, the correlation decreases until one column contains the same constant everywhere, implying zero correlation between them.

Figure 8 shows the results for a dependence SC. We can see that as more errors were generated, the τ value decreased, i.e., the columns became more independent. With

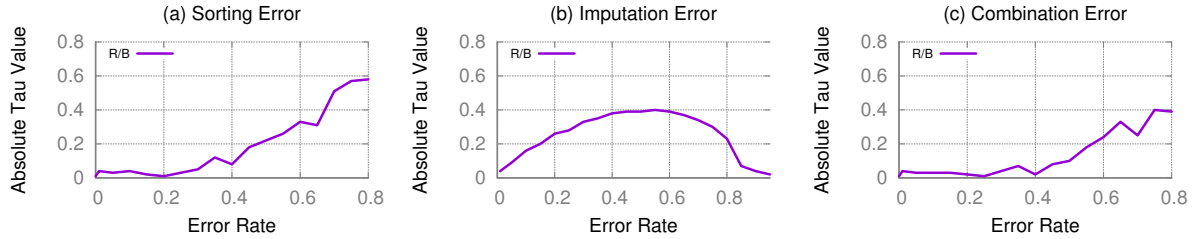

Figure 7: Different error impact on independence SCs (Boston dataset)

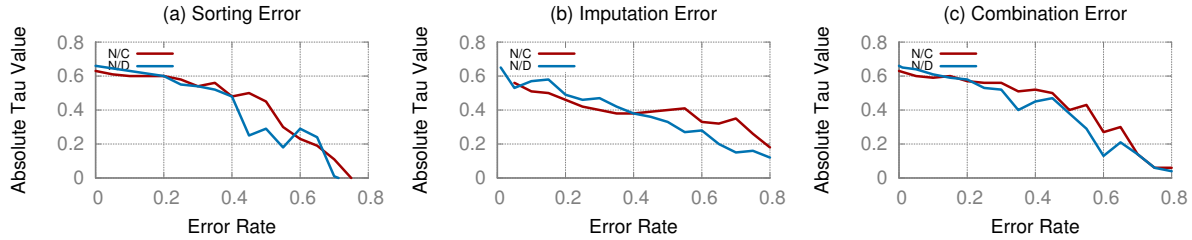

Figure 8: Different error impacts on dependence SCs (Boston dataset)

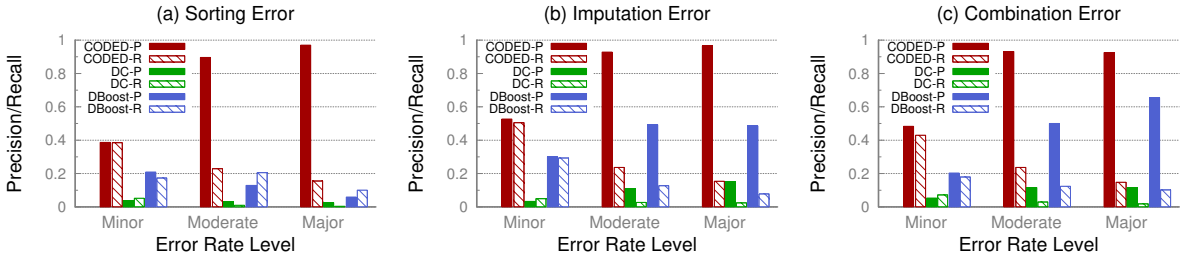

Figure 9: Effectiveness of error detection methods for dependence SCs (Boston dataset, $K = 50$)

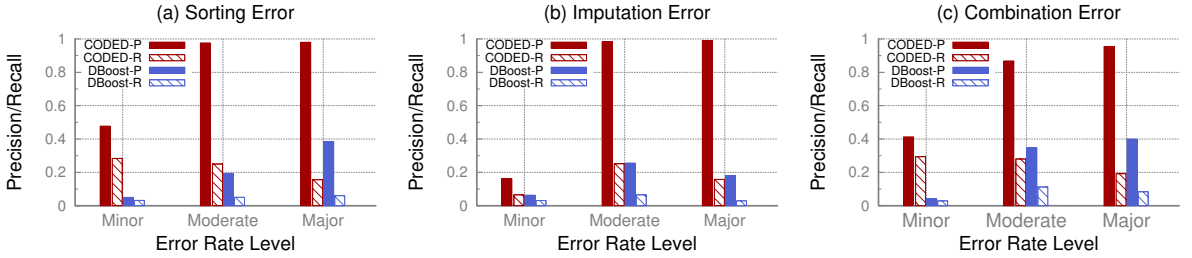

Figure 10: Effectiveness of error detection methods for independence SCs (Boston dataset, $K = 50$)

the error rate at 0.8, the τ value approached zero, indicating an independence relationship.

Exp-3: Evaluation of Error-Detection Approaches. We compared CODED with existing error-detection methods on the Boston dataset. For dependence SCs, we compared with DC and DBoost; for independence SCs, since DC cannot express independence relationships, we compared only with DBoost. We considered all four common forms of SCs: (1) $R \perp\!\!\!\perp B$; (2)$N \not\perp\!\!\!\perp D$; (3) $N \perp\!\!\!\perp B \mid T$; (4) $T \not\perp\!\!\!\perp B \mid C$, as summarized in Table 3.

$R \perp\!\!\!\perp B$ and $N \not\perp\!\!\!\perp D$. Figure 9 and Figure 10 show the results for $R \perp\!\!\!\perp B$ and $N \not\perp\!\!\!\perp D$, respectively. We consider three error levels, depending on the average error rate: *minor error* $= 1\% - 20\%$, *moderate error* $= 20\% - 45\%$, and *major error* $= 50\% - 80\%$. We set $k = 50$, and reported the corresponding precision and recall in these figures.

We first examine CODED's performance. CODED's precision increases with the error level. The reason is that when there is a larger portion of errors, the degree of dependence/independence changes more, thus it is easier for CODED to detect violations. In terms of recall, CODED's decreases with the error level. This is because that we set $k = 50$ for all error levels. In terms of error types, CODED performed averagely good across all error types.

We next compared the performance of CODED, DC and DBoost. As shown in Figure 9 and Figure 10, CODED outperformed the other two approaches. DC did not perform well because the specified denial constraint (i.e., if $r_1[N] > r_2[N]$, then $r_1[D] < r_2[D]$) did not always hold, which led to many false positives. DBoost calculated correlations from dirty data, and then leveraged the calculated relationships to detect errors. However, the errors in the data led to errors in the estimated correlations. In comparison, CODED employs SCs specified by the user, not estimated from the data.

To compare CODED, DC and DBoost succinctly, we reported their F-score (the harmonic mean of precision and

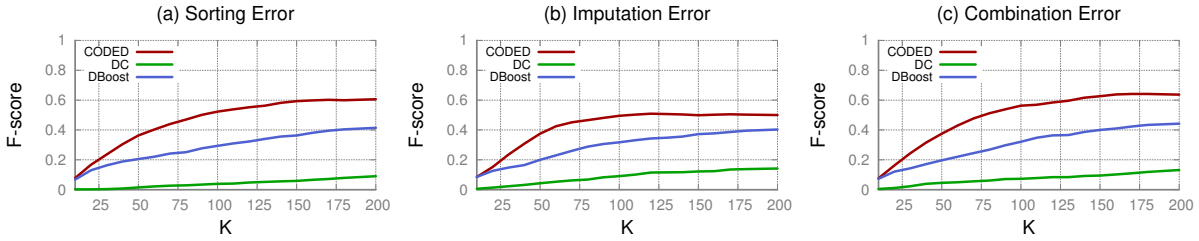

Figure 11: Effectiveness of error detection methods for dependence SCs by varying k (Boston dataset)

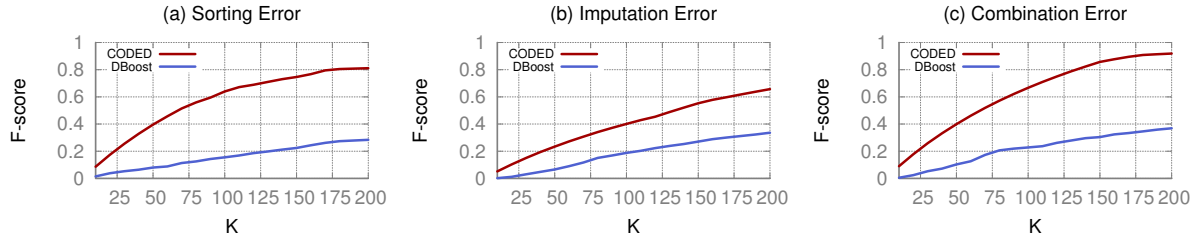

Figure 12: Effectiveness of error detection methods for independence SCs by varying k (Boston dataset)

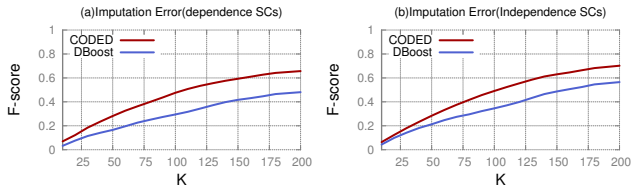

Figure 13: Effectiveness of error detection methods for categorical data (Car dataset)

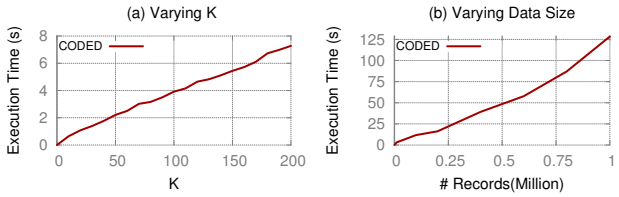

Figure 14: Scalability of CODED (Boston dataset)

recall) under different K values (moderate error level). Results are shown in Figure 11 and Figure 12 for $H \perp\!\!\!\perp B$ and $N \not\perp\!\!\!\perp D$, respectively. We can see that CODED achieved significant higher F-score than DC and DBoost for all settings. However, we also notice that CODED's performance is not as stable as the other two approaches across different error types. CODED performed better for sorting error and combination error, where the average F-score is 0.6 and the max f-score is around 0.8. But for imputation error, the average F-score and the max F-score decrease to 0.5 and 0.6, respectively. As we explained above, if the errors have a small impact on SCs, the power of using SCs to detect the error decreases.

$N \perp\!\!\!\perp B \mid T$ and $T \not\perp\!\!\!\perp B \mid C$. We also examined the effectiveness of CODED for *conditional* independence and conditional dependencies, with moderate error level. The results are very similar to non-conditional cases (see Appendix for more detail).

Exp-4: Effectiveness on Categorical Data. So far, we have only focused on numerical data using the τ test. Next, we use the χ^2 test and evaluate the effectiveness of CODED on categorical data. We selected two SCs ($BP \not\perp\!\!\!\perp Cl$ and $SA \perp\!\!\!\perp DR$) on the Car dataset. DC is not applicable here because there are too many violations for the feasible DCs that we constructed. We compared the performance of CODED and DBoost at the moderate error level. Figure 13 shows the results. Due to the space limit, we just focus on the imputation errors in this experiment. The average F-score of CODED and DBoost are 0.45 and 0.24, respectively. Similar to exp-3, CODED outperforms DBoost in terms of f-score in both independence SC and dependence SC.

Exp-5: Scalability. We evaluate the scalability of CODED in terms of error drill down. We replicated the Boston dataset to enlarge its data size, and chose a dependence SC $N \not\perp\!\!\!\perp D$. We examined the performance of CODED by varying k and n (# of records). The results are shown in Figure 14.

In Figure 14(a), we set # of records to 10,000, and reported the execution time by varying k from 0 to 200. It can be found that the execution time grows linearly. Recall that the time complexity of the K strategy is $\mathcal{O}(n \log n)$ for initialization, and is $\mathcal{O}(kn \log n)$ for selecting k records. The results are consistent with the complexity analysis, and demonstrates the scalability of CODED w.r.t. k.

In Figure 14(b), we fix $k = 50$, and explore the scalability of CODED with different data sizes. The computation time of CODED scales well w.r.t. data size. CODED took less than 1 min (60 seconds) when there are 0.6 million records, and took around 2 min where there are 1 million records.

Exp-6: Effectiveness with Real-life Errors. We evaluate the performance of CODED on the datasets with real-life errors.

Sensor Dataset. Neighboring sensors tend to report similar temperatures, which means that their reports of the temperature should be dependent. The constraint is summarized in Table 6. The results are shown in Figure 15.

The average precision of CODED, DC, and DBoost were 0.93, 0.65 and 0.47, respectively. Unlike the other two methods, the precision of CODED increased continuously, and maintained a high precision around 0.95. For Recall and F-score, all three approaches increased continuously. CODED outperformed the other two methods with a higher increasing speed. Recall that the dataset has two types of errors. We conclude that CODED has the potential ability to detect diverse error types.

Hockey Dataset. For the *Hockey* dataset, the Goal plus-minus column contains many missing values from 1998 to

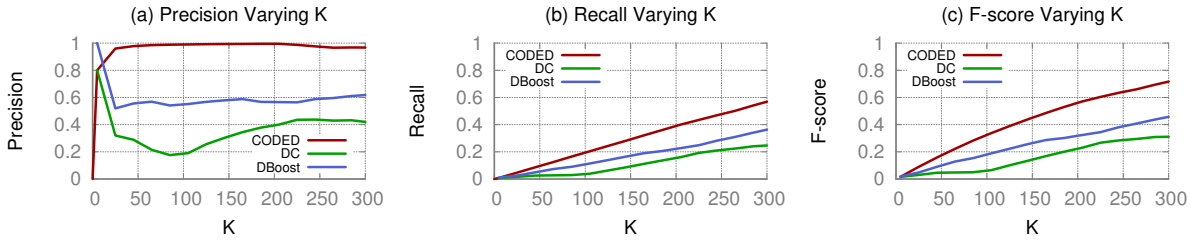

Figure 15: Detecting real-world errors on the Sensor dataset

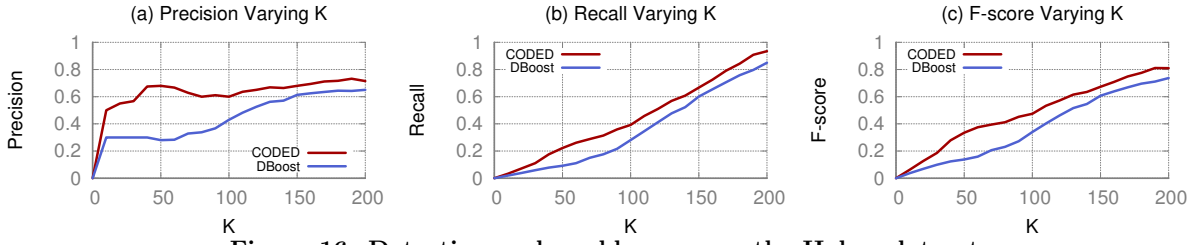

Figure 16: Detecting real-world errors on the Hokey dataset

2004. The curators of the dataset imputed 0 to replace the missing entries in order to build machine learning models. From domain knowledge, we know that the columns *Games(G)* and *Goal Plus-Minus(GPM)* should be independent given *Draft Year(Y)*, because the total number of professional games played by a player is independent of their Plus-Minus before they joined the professional league. However, the imputations made these two columns more dependent. We used CODED and DBoost to detect these errors. The results are shown in Figure 16. We can see that CODED outperformed DBoost in terms of precision, recall, and F-Score. In particular, when k = 50, DBoost only got a precision of 0.28, but the precision of CODED was 0.68 (around 2.5 ×) higher.

7. RELATED WORK

There are many row-oriented constraints proposed for error detection (see [24] for a survey). Unlike these works, SCs are column-oriented constraints (i.e., treat each column as a random variable). Outlier detection often enforces a column-oriented constraint on a single column (e.g., any datapoint that is more than 3 standard deviation is an outlier) [20]. There are also some outlier detection approaches that leverage the relationships between columns to detect outliers [37, 31, 15]. However, none of them allows the user to specify a set of SCs explicitly and then guides the user to detect the errors based on the SCs.

As discussed in Section 2.3, SCs, and ICs share some underlying intuitions, thus it is not surprising to see that some of the ideas in SCs have been explored in the context of ICs. For example, the multi-valued dependencies [17] can be used to express independence relationships; approximate and soft FDs [25, 22, 23, 33] were proposed to relax the hard constraint of FDs and make them tolerate exceptions. However, these methods do not leverage the well-studied hypothesis testing method to detect violations, which complements well downstream statistical modeling. Our approach therefore can be naturally combined with approaches that apply statistics, data mining and machine learning to data cleaning, which have recently attracted significant attention [28, 27, 29, 16, 36, 32, 6, 35, 45, 11, 14, 5]. The SampleClean project studies how to use statistical inference to obtain reliable answers from dirty data [28, 27, 29]. The Holoclean project builds a statistical inference engine to impute, clean, and enrich data [36]. Single-Column detection methods are also proposed [21, 46] to detect data errors and data types for one single column using external resources. Different from the existing works, CODED aims to leverage user-specified statistical constraints.

This paper is focused on constraint-based error detection approaches. There are some other kinds of error detection approaches such as pattern enforcement and deduplication algorithms [3]. It is an interesting future direction to study how to combine different approaches.

There are also some works on automatic IC discovery [10, 9, 44, 30, 12]. In this paper, we assume that SCs are provided by domain experts. There might be some ways to facilitate this process. For example, we can learn a Bayesian Network from the given data, and then automatically generate SCs based on the network. We defer this study to future work.

Error explanation, which aims to provide intuitive explanations to the user about the errors, is another hot topic in data cleaning [43, 41, 40]. In this paper, we returns the top-k erroneous records to help the user understand why an SCs violation happens. We will investigate more advanced techniques (e.g., visualization [43] and Bayesian analysis [40]) in this aspect.

8. CONCLUSION AND FUTURE WORK

This paper proposed a new class of constraints for error detection, called statistical constraints (SCs). Unlike traditional integrity constraints (ICs), SCs treated each column as a random variable and enforced the (in)dependence relationship between random variables. We discussed the advantages of SCs over ICs, and identified the challenges to build an error detection system based on SCs. We proposed a new inference system for SCs: we proved its soundness, and conjectured it to be complete. We developed efficient algorithms for the consistency and implication problems. We discussed how to detect SC violations using hypothesis testing methods for both numerical and categorical data. We developed an error-drill-down framework, and devise efficient top-k algorithms for the framework. We conducted extensive experiments on both synthetic and real-world datasets. The results showed that (1) SCs were effective in detecting errors that violate them compared to DCs and DBoost; (2)

the consistency algorithm was very efficient; (3) the top-k algorithm was scalable w.r.t. k and data size.

Our work represents an initial approach to use SCs for error detection. In addition to future directions mentioned in the previous section, other important topics include the following. (1) *Extensions.* Utilize SCs with more than three variables, more data types, and more independence statistics. (2) *Human-in-the-Loop.* Help the user to discover and validate SCs more efficiently. (3). *Error Repair.* Extend CODED to the error-repairing stage, to automatically repair errors so that the cleaned data satisfies a set of given SCs. Statistical constraints represent a novel approach to leverage powerful statistical methods for error detection. It has great potential to be useful in practice, and opens a new set of research directions in the intersection of statistics and data management.

9. REFERENCES

[1] The Four V's of Big Data. Accessed: 2018-02-28.

[2] Working With Data and Machine Learning in Advertising. `https://soundcloud.com/talkingmachines/episode-thirteen-working-with-data-and-machine-learning-in-advertising`. Accessed: 2018-02-28.

[3] Z. Abedjan, X. Chu, D. Deng, R. C. Fernandez, I. F. Ilyas, M. Ouzzani, P. Papotti, M. Stonebraker, and N. Tang. Detecting data errors: Where are we and what needs to be done? *Proceedings of the VLDB Endowment*, 9(12):993–1004, 2016.

[4] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of databases: the logical level*. Addison-Wesley Longman Publishing Co., Inc., 1995.

[5] L. Akoglu, H. Tong, J. Vreeken, and C. Faloutsos. Fast and reliable anomaly detection in categorical data. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 415–424. ACM, 2012.

[6] L. Berti-Équille, T. Dasu, and D. Srivastava. Discovery of complex glitch patterns: A novel approach to quantitative data cleaning. In *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*, pages 733–744, 2011.

[7] P. Bohannon, W. Fan, M. Flaster, and R. Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 143–154. ACM, 2005.

[8] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for data cleaning. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 746–755. IEEE, 2007.

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

[10] F. Chiang and R. J. Miller. A unified model for data and constraint repair. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 446–457. IEEE, 2011.

[11] X. Chu, I. F. Ilyas, S. Krishnan, and J. Wang. Data cleaning: Overview and emerging challenges. In *Proceedings of the 2016 International Conference on Management of Data*, pages 2201–2206. ACM, 2016.

[12] X. Chu, I. F. Ilyas, and P. Papotti. Discovering denial constraints. *Proceedings of the VLDB Endowment*, 6(13):1498–1509, 2013.

[13] X. Chu, I. F. Ilyas, and P. Papotti. Holistic data cleaning: Putting violations into context. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pages 458–469. IEEE, 2013.

[14] K. Das, J. Schneider, and D. B. Neill. Anomaly pattern detection in categorical datasets. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 169–176. ACM, 2008.

[15] K. Das and J. G. Schneider. Detecting anomalous records in categorical datasets. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Jose, California, USA, August 12-15, 2007*, pages 220–229, 2007.

[16] T. Dasu and J. M. Loh. Statistical distortion: Consequences of data cleaning. *Proceedings of the VLDB Endowment*, 5(11):1674–1683, 2012.

[17] R. Fagin. Multivalued dependencies and a new normal form for relational databases. *ACM Transactions on Database Systems (TODS)*, 2(3):262–278, 1977.

[18] D. Geiger and J. Pearl. Logical and algorithmic properties of conditional independence and graphical models. *The Annals of Statistics*, pages 2001–2021, 1993.

[19] D. Harrison and D. L. Rubinfeld. Hedonic housing prices and the demand for clean air. *Journal of environmental economics and management*, 5(1):81–102, 1978.

[20] J. M. Hellerstein. Quantitative data cleaning for large databases. *United Nations Economic Commission for Europe (UNECE)*, 2008.

[21] Z. Huang and Y. He. Auto-detect: Data-driven error detection in tables.

[22] Y. Huhtala, J. Karkkainen, P. Porkka, and H. Toivonen. Efficient discovery of functional and approximate dependencies using partitions. In *Data Engineering, 1998. Proceedings., 14th International Conference on*, pages 392–401. IEEE, 1998.

[23] Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen. Tane: An efficient algorithm for discovering functional and approximate dependencies. *The computer journal*, 42(2):100–111, 1999.

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

[25] I. F. Ilyas, V. Markl, P. Haas, P. Brown, and A. Aboulnaga. Cords: automatic discovery of correlations and soft functional dependencies. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 647–658. ACM, 2004.

[26] S. R. Jeffery, G. Alonso, M. J. Franklin, W. Hong, and J. Widom. Declarative support for sensor data cleaning. In *International Conference on Pervasive Computing*, pages 83–100. Springer, 2006.

[27] S. Krishnan, M. J. Franklin, K. Goldberg, J. Wang, and E. Wu. Activeclean: An interactive data cleaning framework for modern machine learning. In *Proceedings of the 2016 International Conference on Management of Data*, pages 2117–2120. ACM, 2016.

[28] S. Krishnan, J. Wang, M. J. Franklin, K. Goldberg, and T. Kraska. Privateclean: Data cleaning and differential privacy. In *Proceedings of the 2016 International Conference on Management of Data*, pages 937–951. ACM, 2016.

[29] S. Krishnan, J. Wang, M. J. Franklin, K. Goldberg, T. Kraska, T. Milo, and E. Wu. Sampleclean: Fast and reliable analytics on dirty data. *IEEE Data Eng. Bull.*, 38(3):59–75, 2015.

[30] S. Lopes, J.-M. Petit, and L. Lakhal. Efficient discovery of functional dependencies and armstrong relations. In *International Conference on Extending Database Technology*, pages 350–364. Springer, 2000.

[31] Z. Mariet, R. Harding, S. Madden, et al. Outlier detection in heterogeneous datasets using automatic tuple expansion. 2016.

[32] C. Mayfield, J. Neville, and S. Prabhakar. Eracer: a database approach for statistical inference and data cleaning. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 75–86. ACM, 2010.

[33] 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. *Proceedings of the VLDB Endowment*, 8(10):1082–1093, 2015.

[34] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Elsevier, 2014.

[35] N. Prokoshyna, J. Szlichta, F. Chiang, R. J. Miller, and D. Srivastava. Combining quantitative and logical data cleaning. *PVLDB*, 9(4):300–311, 2015.

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

[37] F. Riahi and O. Schulte. Model-based outlier detection for object-relational data. In *Computational Intelligence, 2015 IEEE Symposium Series on*, pages 1590–1598. IEEE, 2015.

[38] M. Studeny. Conditional independence relations have no finite complete characterization. 1990.

[39] J. Wang and N. Tang. Towards dependable data repairing with fixing rules. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 457–468. ACM, 2014.

[40] X. Wang, X. L. Dong, and A. Meliou. Data x-ray: A diagnostic tool for data errors. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 1231–1245, 2015.

[41] X. Wang, A. Meliou, and E. Wu. Qfix: Diagnosing errors

through query histories. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1369–1384. ACM, 2017.

[42] L. Wasserman. *All of statistics: a concise course in statistical inference*. Springer Science & Business Media, 2013.

[43] E. Wu and S. Madden. Scorpion: Explaining away outliers in aggregate queries. *Proceedings of the VLDB Endowment*, 6(8):553–564, 2013.

[44] C. Wyss, C. Giannella, and E. Robertson. Fastfds: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances extended abstract. In *International Conference on Data Warehousing and Knowledge Discovery*, pages 101–110. Springer, 2001.

[45] M. Yakout, L. Berti-Équille, and A. K. Elmagarmid. Don't be scared: use scalable automatic repairing with maximal likelihood and bounded changes. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 553–564. ACM, 2013.

[46] C. Yan and Y. He. Synthesizing type-detection logic for rich semantic data types using open-source code. In *Proceedings of the 2018 ACM SIGMOD International Conference on Management of data*. ACM, 2018.

APPENDIX

A. PROOF OF LEMMA

PROOF 1 (PROOF OF LEMMA 1). *The core idea of this proof is to prove that by applying the $\mathcal{G}$. Given a set of independence* SCs $S = \{\mathrm{SC}_1, \mathrm{SC}_2, ..\mathrm{SC}_n\}$,
(1). We first prove SCs $S^*_{\mathcal{A}}$ *entailed by S using $\mathcal{A}$ and* SCs $S^*_{\mathcal{G}}$ *entailed by S using $\mathcal{G}$ satisfy the relationship: $S^*_{\mathcal{G}} \subseteq S^*_{\mathcal{A}}$. As A covers all the* SCs *that G can infer at each iteration. Obviously, $S^*_{\mathcal{G}} \subseteq S^*_{\mathcal{A}}$.*
(2). We next prove new SCs $S^*_{\mathcal{G}}$ *entailed by S using $\mathcal{G}$ and new* SCs $S^*_{\mathcal{A}}$ *entailed by S using $\mathcal{G}$ satisfy the relationship: $S^*_{\mathcal{A}} \subseteq S^*_{\mathcal{G}}$.*

To prove this, we first define a sequence of applying $\mathcal{G}$ and $\mathcal{A}$.

$\mathcal{A}$ contains four axioms: Symmetry(a), Decomposition(b), Weak union(c) and Contraction(d). In this paper, we limit the form of SC *to be in either $A \perp\!\!\!\perp (\not\perp\!\!\!\perp) B$ or $A \perp\!\!\!\perp (\not\perp\!\!\!\perp) B \mid C$. Obviously, if there expects to have more* SCs *entailed by given* SCs*, we need to apply Contraction axiom first. The Decomposition axiom and Weak Union axiom can bring new* SCs *independently. Therefore, the sequence of applying the axioms become (b) $\rightarrow$ (c)/(d)(we omit (a) here as (a) didn't essentially bring new* SCs*).*

We also find that given a SCs *set in the form of $A \perp\!\!\!\perp (\not\perp\!\!\!\perp) B, A \perp\!\!\!\perp (\not\perp\!\!\!\perp) B \mid C, A \perp\!\!\!\perp BC$, no more new* SC *can be entailed by $A \perp\!\!\!\perp BC$ (Conclusion 1). It is trivial to prove this. It is obviously to see by $A \perp\!\!\!\perp BC$ can never satisfy the sequence of inferring more* SCs *with $A \perp\!\!\!\perp BC$. Hence, no more new* SCs *entailed by $A \perp\!\!\!\perp BC$ can be found.*

Following the sequence to compute $S^*_{\mathcal{A}}$, there are two situations describing the relationship between given two initial SCs*,*

(I1) There are no SCs *satisfies Contraction axioms and the sequence cannot start. Obviously, $S^*_{\mathcal{G}} = S^*_{\mathcal{A}} = S$. Hence, the conclusion $S^*_{\mathcal{G}} \subseteq S^*_{\mathcal{A}}$ holds.*

(I2) two SCs *satisfies Contraction axioms should be matched. So at least there exists two* SCs *in the form: $A \perp\!\!\!\perp B$ or $A \perp\!\!\!\perp C \mid B$. For the two* SCs*, we follow the implication sequence, and apply (d) first. Then we get one new $SC^1_{new} = A \perp\!\!\!\perp BC$. The next step is two applying both (b) and c. By applying (c) and d, we get $SC^2_{new} = A \perp\!\!\!\perp B \mid C$ and $SC^3_{new} = A \perp\!\!\!\perp C$. As we expect* SCs *to be in three variable setting, we can draw the conclusion from (Conclusion 1) and delete $SC^1_{new} = A \perp\!\!\!\perp BC$. Apparently, two* SCs *can be covered by applying A. And the process can be repeated sequentially till no more new* SCs *entailed by the given set can be found. At each step i, $S^{(i)*}_{\mathcal{A}} \subseteq S^{(i)^*}_{\mathcal{G}y}$. Therefore, $S^*_{\mathcal{A}} \subseteq S^*_{\mathcal{G}}$.*

Combing (1) and (2), we can have $S^*_{\mathcal{G}} = S^*_{\mathcal{A}}$. Hence Lemma 1 is proved.

PROOF 2 (PROOF OF LEMMA 2). *To prove this, we first prove given two independence* SCs $S = \{\mathrm{SC}_1, \mathrm{SC}_2\}$*, the involving variable set $V = \{v_1, v_2, .., v_n\}$ w.r.t. S. There are following two situations describing the relationship between* SC_1 *and* SC_2*.*
(1). SC_1 *and* SC_2 *don't satisfy the condition of $\mathcal{G}$. That means no more* SC *can be entailed by S. Obviously, no more variables beyond V will be added;*
(2). SC_1 *and* SC_2 *satisfy the condition of $\mathcal{G}$. That means, they can be rewritten as $S = \{A \perp\!\!\!\perp B, A \perp\!\!\!\perp C|B\}$ and $V = \{A, B, C\}$. Therefore, we apply $\mathcal{I}$, we have $S^{'} = \{A \perp\!\!\!\perp B, A \perp\!\!\!\perp C, A \perp\!\!\!\perp C|B, A \perp\!\!\!\perp B|C\}$ and $V^{'} = \{A, B, C\}$. Obviously, $V = V^{'}$.*

Combing (1) and (2), we can prove that after applying $\mathcal{I}$, no more variables beyond the original V will be generated. This implies for one single SC*, it only generate new* SC *within the variables it and its matching* SC*.*

Furthermore, we apply the combination theory. As no more variables between where there are $2\binom{3}{2}$ possible outcomes. And for a single SC *it can at most generate* 6 SC*. Therefore, it is proved that I^* implied by I at most contains $6|I|$ number of* SCs*, which means $|I^*| \leq 6|I|$.*

B. EFFICIENT IMPLEMENTATION FOR K STRATEGY

We discuss the efficient Implementation of the K strategy detailed here. As illustrated in Section 5.2, we choose another data structure *segment tree* in the initialization phase so as to greatly reduce the time complexity. Before applying this idea, some pre-processing is needed. We first sort dataset $D =< x_1, y_1 >, ... < x_n, y_n >$ by X column. Then we scan the new data set to obtain the concordant pairs, and non-concordant pairs. Later, we insert a segement tree $[y_i, y_i]$ to the segment tree. After the initialization phase, we continue do the iterations as shown in the Algorithm 2. Algorithm 3 illustrates the pseudo-code. This efficient implementation is also applicable to K^c strategy.

C. EXPERIMENTAL ANALYSIS FOR CONDITIONAL SCs

$N \perp\!\!\!\perp B \mid T$ and $T \not\perp\!\!\!\perp B \mid C$. We also examined the effectiveness of CODED for *conditional* independence and conditional dependencies, with moderate error level. Constraints used for CODED, DC and DBoost are summarized in Table 6. To accommodate the variables of constraints to CODED framework, we further bucket the value of *Tax Rate* into categorical values. We categorize the numerical value into 6 buckets following its distribution. We reported their F-score under different K values at moderate error level, which is similar to Exp-3. Results are shown in Figure 17 and Figure 18 respectively. CODED obtained a higher F-score than another two state-of-art methods, which is even more significant under independence SCs. Also, similar to Exp-3 and Exp-4, the detection effectiveness of imputation error is not as stable as that of another two types errors. The average F-score of CODED under independence SCs is 0.55 and with

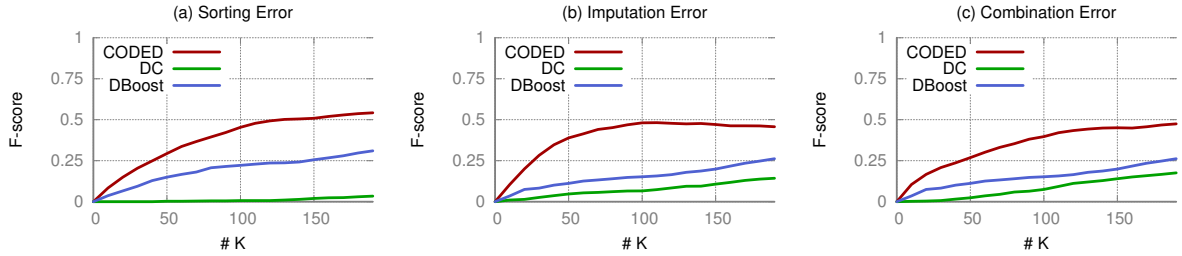

Figure 17: Effectiveness of error detection methods for conditional dependence SCs (Boston dataset, $K = 50$)

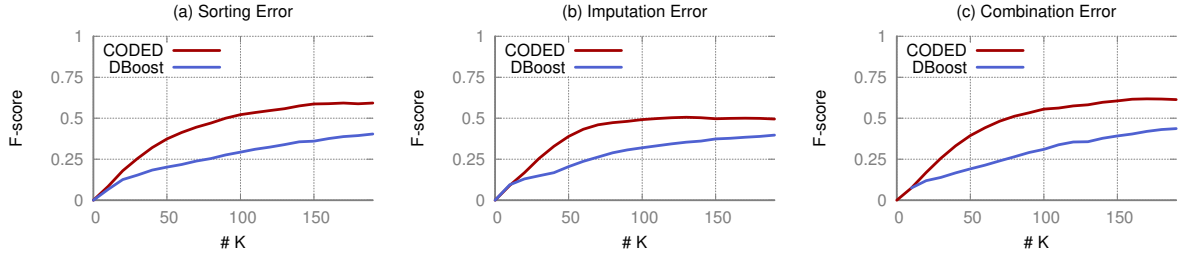

Figure 18: Effectiveness of error detection methods for conditional independence SCs (Boston dataset, $K = 50$)

Algorithm 3: Efficient τ-test-based error detection algorithm

Input: An SC $= X \perp\!\!\!\perp (\not\perp) Y$, Dataset $D = \{< x_1, y_1 >, ... < x_n, y_n >\}$, k
Output: k records

$T \leftarrow \emptyset$; //Segment Tree
$Q \leftarrow \emptyset$; // Priority Queue
$R \leftarrow \emptyset$; // Returned List
benefit$(< x_i, y_i >) = 0$ for $< x_i, y_i > \in D$;
Sort D by X column value ;
// Initialization
for $< x_i, y_i > \in D$ **do**
    $n_c = T.query([-\infty, y_i])$;
    $n_d = T.query([y_i, +\infty])$;
    benefit$(< x_i, y_i >) = 2n_c + (|D| - n_c - n_d)$;
    $T.insert([y_i, y_i])$;
    $Q.push(< x_i, y_i >,$ benefit$(< x_i, y_i >)$;
// Iteration
for $i = 1$ *to* k **do**
    Add $Q.top()$ to R;
    Update Q; // Update the weight of each record benefit$(< x_i, y_i >) \in Q$ by querying the segment tree T
return R;

max F-score 0.69, however the two results will decrease to 0.43 and 0.51 with imputation error inserted. This again demonstrates that the CODED can detect the power of errors as if the power is not as significant, CODED is not that powerful.