Discovery of Complex Glitch Patterns: A Novel Approach to Quantitative Data Cleaning

Laure Berti-Équille[1], Tamraparni Dasu[2], Divesh Srivastava[2]

[1]*University of Rennes 1*
Campus de Beaulieu, Rennes, France
` cpr g>gpgq_, dp
[2]*AT&T Labs - Research*
Florham Park, NJ 07932, USA
&r_kp*bgt cqf' >pcqc_paf, _rr, amk

***Abstract*— Quantitative Data Cleaning (QDC) is the use of statistical and other analytical techniques to detect, quantify, and correct data quality problems (or glitches). Current QDC approaches focus on addressing each category of data glitch individually. However, in real-world data, different types of data glitches co-occur in complex patterns. These patterns and interactions between glitches offer valuable clues for developing effective domain-specifi quantitative cleaning strategies.**

In this paper, we address the shortcomings of the extant QDC methods by proposing a novel framework, the DEC (*Detect-Explore-Clean*) framework. It is a comprehensive approach for the definition detection and cleaning of complex, multi-type data glitches. We exploit the distributions and interactions of different types of glitches to develop data-driven cleaning strategies that may offer significan advantages over *blind* strategies. The DEC framework is a statistically rigorous methodology for evaluating and scoring glitches and selecting the quantitative cleaning strategies that result in cleaned data sets that are *statistically proximal* to user specifications We demonstrate the efficacy and scalability of the DEC framework on very large real-world and synthetic data sets.

I. Introduction

As data types and data formats change to keep up with evolving technologies and applications, data quality problems (or glitches) too have evolved and become more complex. Data streams, Web logs, Wikipedias, biomedical applications, and social networking websites generate a mind boggling variety of data types with attendant data quality problems [1]. In real-world data, different types of glitches co-occur in complex patterns such as the simultaneous occurrence of outliers and missing values in the same record.

In the last two decades, a large body of work in the DB community has focused on declarative data cleaning and more recently, on constraint-based data repair [2], but very few approaches have considered the issues of quantitative data cleaning [8], [3]. Current state of the art in quantitative data cleaning approaches has certain limitations:

• **One-dimensional approach:** Current statistical and data mining techniques address each category of data glitch in isolation during data preparation and data shaping process [11]. In real-world data, glitches co-occur across multiple attributes and records with complex interactions. The dependence could arise through a common root cause, or due to the fact that some data are derived from other parts of the data set that could be potentially fl wed, leading to *correlated data quality problems*. Such one-at-a-time approaches could impair the detection and cleaning of other types of glitches, e.g., treating missing values through imputation could mask duplicates.

• **Unrealistic assumptions:** In addition to treating glitches as simple uncorrelated entities, existing methods often make unrealistic assumptions about the glitch distributions, like Missing At Random (MAR) or that they constitute a small portion of the data. In reality, although some glitches may occur randomly, different types of glitches often have complex inter-dependencies and a significan portion of the data could be impacted by the glitches. Detecting and analyzing the patterns of glitches provide valuable clues to developing effective cleaning strategies.

• **De-linking cleaning and detection:** Often, the quantitative treatment of glitches is pre-determined (e.g., impute missing values using median) and has no connection to other types of glitches that are detected. Or, multiple methods for detecting the same glitch could result in conflictin results. A one-dimensional cleaning strategy does not take such conflictin results into consideration. For example, a boxplot method might declare a value an outlier, but confidenc regions using joint distributions might not fin the observation to be an outlier. Or, cleaning a glitch (e.g., de-duplication) could result in a new glitch (e.g., outlier). Therefore, in principle, the "cleaned" data could be of poorer quality than the original "dirty" data. These issues go unaddressed when there is no connection between cleaning and detection of glitch types.

A. Contributions

In this paper, we propose a framework that identifie and exploits complex glitch patterns for data cleaning and develop strategies that outperform classical QDC strategies. Our contributions include:

• **Definition and methodology for detecting and treating complex glitches:** We consider data glitches to be complex interconnected entities, and that their distributions and interactions hold important clues to adequate cleaning strategies. We introduce and defin the notions of *multi-type*, *multi-occurrent*, and *concomitant data glitches*, *glitch signatures* and

TABLE I EXAMPLE OF SNMP DATA SET AND POSSIBLE QUANTITATIVE CLEANING STRATEGIES

(a) SNMP Data Set

poll#	date-time	ID	IN	OUT
1	09/01/2009-00:56:00	1	-	-
2	09/01/2009-01:01:00	1	120	130
3	09/01/2009-01:01:00	1	130	150
4	09/01/2009-01:05:00	1	-	560
5	09/01/2009-01:05:00	1	-	-
6	09/01/2009-01:10:00	1	140	140
7	09/01/2009-01:10:00	1	110	100
8	09/01/2009-00:56:00	2	130	110
9	09/01/2009-01:01:00	2	-	256
10	09/01/2009-01:01:00	2	-	-
11	09/01/2009-01:05:00	2	120	130
12	09/01/2009-01:05:00	2	130	120
	Sum		880	1696
	Median		130	130

(b) Strategy 1

Deletion of records with either missing or inconsistent values followed by deletion of duplicate records that do not preserve the original median

3	09/01/2009-01:01:00	1	130	150
6	09/01/2009-01:10:00	1	140	140
8	09/01/2009-00:56:00	2	130	110
12	09/01/2009-01:05:00	2	130	120
	Sum		530	520
	Median		130	130

(c) Strategy 2

Replacement of missing values by the original median followed by deletion of the duplicate records that *(i) contain inconsistencies and (ii) do not preserve the original median*

1	09/01/2009-00:56:00	1	130	130
2	09/01/2009-01:01:00	1	120	130
5	09/01/2009-01:05:00	1	130	130
6	09/01/2009-01:10:00	1	140	140
8	09/01/2009-00:56:00	2	130	110
10	09/01/2009-01:01:00	2	130	130
11	09/01/2009-01:05:00	2	120	130
	Sum		900	900
	Median		130	130

(d) Strategy 3

Based on the discovered patterns: Deletion of fully missing records and records containing inconsistencies and replacement of timestamps

2	09/01/2009-00:56:00	1	120	130
3	09/01/2009-01:01:00	1	130	150
6	09/01/2009-01:05:00	1	140	140
7	09/01/2009-01:10:00	1	110	100
8	09/01/2009-00:56:00	2	130	110
11	09/01/2009-01:01:00	2	120	130
12	09/01/2009-01:05:00	2	130	120
	Sum		880	880
	Median		130	130

glitch scores, and describe their computation (Section III). We employ statistical tests to discover the joint distribution and interaction of different types of glitches, and we propose the GPD (Glitch Pattern Discovery) algorithm and use it to discover *glitch patterns* (Section IV).

• **Process for linking detection and cleaning:** The patterns, when explained by domain experts, result in the development of *effective knowledge-based cleaning strategies* that could potentially lead to the *elimination of the root cause of the glitches*. Thus, glitch pattern exploration is an important way of linking detection and cleaning. We call this the *Detect-Explore-Clean* (DEC) process (Section II). In addition to developing domain specifi solutions, this process allows experts to determine whether the quantitative cleaning strategies are meaningful in the context of the application domain.

• **Framework for findin the best cleaning strategies:** We introduce the notions of *cost* and *effectiveness* of a quantitative cleaning strategy, and the framework to identify *best* strategies that result in cleaned data sets that are *statistically proximal* (*i.e*, the closest) to a user-specifie "ideal" data set (Section V).

• **Validation:** We demonstrate the accuracy and scalability of the DEC framework on real-world and synthetic data sets (Section VI).

B. Illustrative Example

Consider the traffic logs collected by network management stations that use SNMP (*Simple Network Management Protocol*). Pollers periodically request measurements from interfaces of network devices; inbound and outbound traffic get polled at both endpoints of each link of the network.

Data quality issues in SNMP data feeds include: 1) missing values due to failures in a network device, unreachable elements, or loss of UDP packets; 2) duplicate values that occur when different pollers are assigned common polling responsibilities due to erroneous or out-of-date configuratio tables; 3) inconsistencies that are caused by violation of network traffic laws, e.g., the sum of traffic transmitted from a device exceeds the capacity of the device. For simplicity, we limit our example to traffic through a unique device with 2 interfaces, identifie by a timestamp and interface ID in Table I(a). Consider the `IN` and `OUT` values in Table I(a). The table contains the following data glitches:

- Missing values for polls 1, 4, 5, 9, and 10,
- Duplicate polls for pairs (2,3), (4,5), (6,7), (9,10), and (11,12) identifie by identical timestamps and IDs,
- Inconsistent `OUT` values in polls 4 and 9 that exceed the network link capacity (> 200) and cause a noticeable disparity between the total inbound traffic rate and outbound traffic rate (880 ≠ 1696).

We observe the following patterns of glitches:

- Co-occurrence of same glitch types: Polls 1, 5 and 10 exhibit (*missing* `IN`, *missing* `OUT`) pairs of values,
- Co-occurrence of multiple glitch types: Polls 4 and 9 contain (*missing* `IN`, *inconsistent* `OUT`) pairs,
- All (*missing*`IN`, *missing* `OUT`) pairs (namely, polls 1, 5 and 10) are followed by two duplicates of (`IN`, `OUT`), namely, pairs of polls (2,3), (6,7) and (11,12).

The example of Table I(a) illustrates that *complex data glitches* are *frequent*, *multi-occurrent* (*i.e.*, the same glitch type occurs multiple times), *concomitant* (*i.e.*, a glitch type co-occurs with other glitch types within the same row), and *multi-type* (*i.e.*, multiple error types may affect the same value/row) and may also have some temporal dependencies. We formalize these notions in Section III-A. The number of quantitative cleaning strategies to clean these glitches is potentially very large due to the combinatorial possibilities of combining and ordering individual quantitative cleaning techniques. Annexe D presents the list of methods for quantitative data cleaning that we use in our experiments.

Tables I(b-d) contain the results of three cleaning strategies.

• **Strategy 1** deletes records with outliers or missing values, and removes duplicates to preserve the original median (130). The resulting data set has only 4 records: 3, 6, 8, and 12 whose inrate and outrate values satisfy the link capacity constraint (≤ 200), but their sum does not satisfy the traffic law (530 ≠ 520);

• **Strategy 2** replaces all missing values by the original median and then deletes duplicate records that contain either inconsistent or outlying values or violate the original median. In addition to excluding polls 3, 4, and 9, the fina result of Strategy 2 excludes polls 7 and 12;

• **Strategy 3** takes advantage of the discovered patterns: a pair of (*missing* `IN`, *missing* `OUT`) is followed by duplicate polls in 3/3 cases. According to the experts in the application domain, this pattern actually identifie a polling delay, and the values of the firs duplicate polls 2, 6, and 11 are actually the preceding polls with missing values. The second and third strategies preserve the original median and satisfy the link capacity constraint and traffic law.

Note that the order in which cleaning methods are applied makes a difference. In order to choose among the three

strategies, we can estimate the cost and effectiveness of the strategies but we need additional criteria to compare the resulting cleaned data sets that best suit the user's requirements.

Suppose the user has a set of specification or an example of an acceptable or *ideal* data set (e.g., "the ideal data set should have at least 90% of the original data set size, preserve median, and satisfy the constraint on link capacity (≤ 200)"). Intuitively, the best strategy will result in the data set that is the closest to this ideal, as measured by a distance measure. In this paper, we propose choosing cleaning strategies that result in data sets that are guaranteed to be within a certain distance of the ideal data sets. We call these *statistically proximal* data sets, to be define in Section V. In the following section, we describe the *Detect-Explain-Clean* process that leverages complex glitch patterns to build smart quantitative cleaning strategies.

II. The DEC Framework

Most data mining methods and data analysis techniques pre-suppose the availability of a data set that meets certain assumptions. For example, that all samples in the data have the same statistical distribution and are *i.i.d*, or that there are no missing values. In real life, data sets are heterogeneous since different parts are often drawn from different sources, and seldom conform to any assumptions. For data mining results to be reliable and accurate, it is imperative that the observed data set **D** be cleaned to match the assumptions embodied in the ideal data set denoted by $\mathbf{D}^\star$. Examples of ideal data sets include a set of constraints that the data should satisfy (e.g., no missing values, less than 2% outliers), or distributional assumptions (e.g., unimodal distribution of data).

Intuitively, the purpose of data cleaning is to transform the original data set **D** into a "cleaner" data set D^S, using a strategy S comprising of a sequence of cleaning methods such that D^S is guaranteed to resemble $\mathbf{D}^\star$ within certain bounds, while meeting certain cost and efficiency requirements.

A. Problem Statement

Formally, the problem can be stated as follows: Find the quantitative cleaning strategy B among candidate strategies $\mathcal{S}$ such that its resulting data set D^B is *proximal* to the ideal data set $\mathbf{D}^\star$ as

$$D^B = \arg\min_{\{S \in \mathcal{S}\}} [\mathtt{dist}(D^S, \mathbf{D}^\star)] \quad \text{subject to } Cost(S) \le \mathbf{U} \text{ and } Eff(S) \ge \mathbf{\Gamma} > 0 \qquad (1)$$

where `dist` is a distance function that measures the distance between two data sets, **U** is a pre-define upper bound for the cost, and $\mathbf{\Gamma}$ is the lower bound of $Eff(S)$, the effectiveness of the cleaning strategy S. These notions will be define in detail in Section V. In this paper, we will demonstrate that quantitative cleaning strategies derived from complex glitch patterns perform better than *blind* quantitative cleaning strategies in terms of generating proximal data sets (*i.e.*, with minimal distance to the ideal data set).

B. Overview of the Approach

To fin a proximal quantitative cleaning strategy we propose the following three steps of the DEC framework:

1) Identify and score the glitches: First, we create a *glitch signature* for every cell (*i.e.*, value) of the data set by applying a suite of glitch detection methods. We resolve conflictin detection responses for the same glitch type by combining the scores computed from various detection methods into a single *glitch score*, noted g_{ij} for each cell. These glitch scores at the individual cell level are further aggregated into $g(\mathbf{D})$, the *aggregate glitch score* for the entire data set.

2) Develop multivariate cleaning strategies: Second, we use statistical tests to identify and explore interactions between glitch categories, and other patterns of association. We use the glitch scores and patterns of glitches to formulate novel, data-driven *multivariate* cleaning strategies to augment the set of conventional quantitative cleaning strategies.

3) Find proximal quantitative cleaning strategies: Finally, we use cost and effectiveness constraints to reduce the set of candidate cleaning strategies. As a fina step, we identify cleaning strategies whose resulting cleaned data sets are closest to the user-define ideal data set.

In this paper, we focus on glitches at the value level, and record level for duplicates. Other levels of granularity, such as data distributions (e.g., to detect *disguised missing values* or incorrect but *default* values [10]) are deferred to future work. Furthermore, identifying optimal strategies (in terms of efficiency and performance) is outside the scope of this paper but will be addressed in the next step of our work.

III. Detecting and Exploring Data Glitches

A. Concomitant and Multi-Occurrent Glitches

Let the observed data set **D** be an $n \times k$ matrix of numerical, ordinal, and categorical data values. Let d_{ij} denote the value of the jth attribute for the ith record.

Let **M** be the set of M methods applied to the data set **D** to detect glitches at the value (cell) level. The set of M glitch detection methods maps each value d_{ij} of the data set **D** to an M-dimensional vector, called a detection signature $\mathbf{g}_{ij} = \{I_{ijh}\}$.

The set of signatures $\mathbf{G} = \{I_{ijh}\}$ is a $n \times kM$ matrix of binary values, where I_{ijh} is an *indicator function* that is equal to 1 if method h detects a glitch in row i and column j, and 0 otherwise.

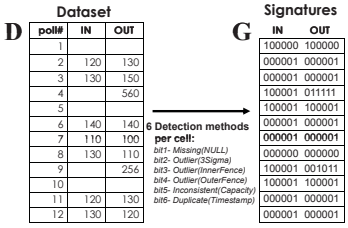


Fig. 1. SNMP data set and its glitch signatures

Example 1: In Fig. 1, {100001011111} is the glitch signature of poll 4. It corresponds to the concatenation of cell signatures of the `IN` and `OUT` values of poll 4. The firs and

seventh bit (from left to right) correspond to missing values, bits 2 through 5 and 8 through 11 correspond to outliers and inconsistencies identifie by different detection techniques, and duplicates are identifie by methods corresponding to bits 6 and 12. All other detection methods (remaining bits) returned null values.

Assume that there are C categories of glitches, $c = 1, \ldots, C$. Each glitch category has at least one detection method associated with it. Each detection method refers to only one category of glitch and acts independently of other detection methods. The number of glitches of type c associated with d_{ij} is given by $m_{ij}(c) = \sum_{h \in \mathbf{M}_c} I_{ijh}$, where $\mathbf{M}_c$ is the set of methods that identify glitches of category c. The total number of glitches associated with value d_{ij} is given by $m_{ij} = \sum_c m_{ij}(c)$ $(1 \le c \le C)$.

We use the notations described above to define

Glitch. A value $d_{ij} \in \mathbf{D}$ is a glitch if its detection signature has at least one non null element, *i.e.* if $m_{ij} > 0$.

Multi-Type Glitch. A value $d_{ij} \in \mathbf{D}$ is a multi-type glitch if there are at least two categories of glitches associated with it: $\exists\ c$ and c' such that $m_{ij}(c) \ge 1$ and $m_{ij}(c') \ge 1$ $(c \ne c')$.

Concomitant Glitches. Two or more values d_{ij} and $d_{ij'}$ $(j \ne j')$ in $\mathbf{D}$ are concomitant glitches if they are both glitches ($m_{ij} > 0$ and $m_{ij'} > 0$ $(j \ne j')$), and occur in the same row i.

Multi-Occurrent Glitch. A multi-occurrent glitch d in the data set $\mathbf{D}$ is a glitch whose detection signature $\mathbf{g}$ is shared by two or more distinct values in the data set: $\exists\ d_{i,j}$ and $d_{i'j'}$, such that $g_{i,j} = g_{i'j'} = \mathbf{g}$.

Example 2: In Fig. 1, at the value level, {100001} is the signature of a multi-occurrent glitch that is associated with 6 values. They share two categories of glitches: missing and duplicate. The signature is concomitant with the glitch signature {011111} for poll 4, with {001011} for poll 9, and with the same signature {100001} for polls 5 and 10.

B. Glitch Scoring

In this section, we introduce the notion of *glitch score*, both at the value level and at the data set level. First, in order to associate a unique score with a given glitch category (e.g., outlier), we combine scores from multiple methods that detect the same type of glitch. This is particularly important for conflic resolution when different detection methods return different responses. Second, we associate a single scalar glitch score with a value $d_{ij} \in \mathbf{D}$ by assigning weights to each category of glitch (characterizing severity or project impact) and computing a weighted average. The weighted glitch score of individual cells is further aggregated across the entire data set to measure the quality of the data set.

Combining Multiple Detection Methods

We use κ, Fleiss' Kappa measure (define in Annexe A) to combine binary responses from multiple independent methods that detect the same type of glitch. κ represents the degree of agreement in detection over and above that expected by chance (See [6] for details). Intuitively, we reinforce the score when several detection methods that generally disagree, provide the same detection response. Conversely, we attenuate the score of the glitch category when the detection methods that generally agree, do not provide the same detection response. We have $\kappa = 1$ when there is complete general agreement of the detection methods, and $\kappa = 0$ when there is complete disagreement. For each category c with M_c detection methods, we compute a glitch score $g_{ij}(c)$ based on the glitch signature and Fleiss' Kappa measure κ_c as follows:

$$g_{ij}(c) = \alpha \frac{m_{ij}(c)}{M_c} \text{ with } \alpha = \begin{cases} 1 & \text{for } M_c = 1 \\ \dfrac{M_c - \kappa_c + \delta_{ij}(c)}{2M_c - 1} & \text{for } M_c > 1 \end{cases} \quad (2)$$

$\delta_{ij}(c)$ is the ratio of number of agreements over the total number of pairs of methods that fla and associate value d_{ij} with glitch category c. The glitch score has the following properties:

Idempotence. The score of a glitch category c returned by only one detection method is identical to the score of a set of detection methods $\mathbf{M_c}$ in general agreement ($\kappa_c = 1$) returning the same detection response for this category.

Reinforcement. If the methods in $\mathbf{M_c}$ generally disagree on their detection responses ($\kappa_c = 0$) but return the same detection response for a given value d_{ij}, the score of the glitch category is greater than the one obtained with general agreement.

Example 3: Consider the OUT values in the example of Fig. 1 with respective signature {0<u>111</u>11} for poll 4 and {00<u>10</u>11} for poll 9. Consider the two following cases where we compute the score for the outlier category detected by three methods (with underlined detection responses):

- If the three methods generally agree ($\kappa_{outlier} = 1$), then the score of the outlier category will be $g_{42}^{\kappa_1}(outlier) = 1$ for the OUT value of poll 4 and $g_{92}^{\kappa_1}(outlier) = 0.37778$ for the OUT value of poll 9.
- If the three methods for detecting outliers generally disagree ($\kappa_{outlier} = 0$), then the score of the outlier category will be $g_{42}^{\kappa_0}(outlier) = 1.2$ and $g_{92}^{\kappa_0}(outlier) = 0.44444$.

$g_{42}^{\kappa_1}(outlier) = 1$ (idempotence) and $g_{42}^{\kappa_0}(outlier) > g_{42}^{\kappa_1}(outlier)$ (reinforcement) illustrate the properties of the glitch score.

Weighting Glitch Categories

The importance of a glitch type varies by application. An application focused on typical values might place only a small penalty on missing values and high penalty on outliers since they distort the computation of typical values. But applications involving rare events might put a big penalty on missing values and a small penalty on outliers. To assign differential priorities to glitches, we defin a *weighted glitch score* where the weight of each glitch category is determined by the user or subject matter expert. The weighted glitch score is given by:

$$g_{ij} = \sum_c w_c\ g_{ij}(c) \text{ where } \sum_c w_c = 1. \quad (3)$$

The *aggregate glitch score* of a data set $\mathbf{D}$ is define as

$$g(\mathbf{D}) = \frac{1}{nk} \sum_{i,j} g_{ij}. \quad (4)$$

Monotonicity property of the aggregate glitch score. Consider two data sets D and D',

- If D and D' have the same size and their respective glitch sets $\mathcal{G}$ and $\mathcal{G}'$ are such as $\mathcal{G} \subseteq \mathcal{G}'$, then $g(D) \leq g(D')$.
- If $D \subseteq D'$ and their respective glitch sets $\mathcal{G}$ and $\mathcal{G}'$ are such as $\mathcal{G} \equiv \mathcal{G}'$, then $g(D') \leq g(D)$.

Example 4: Consider the SNMP data set in Table I(a) where $\kappa_{outlier} = 0.86447$, and assume equal weights assigned to all glitch categories. Then, $g(\mathbf{D}) = 0.32723$. If we replace 5 existing values with missing values, the glitch score of the new data set is $g(\mathbf{D}^1) = 0.379312 > g(\mathbf{D})$. If we remove 3 duplicate polls (2, 5, and 12) and consequently exclude 6 values from the original data set, the new glitch score is $g(\mathbf{D}^2) = 0.32519 < g(\mathbf{D})$. Finally, if we remove poll 8, the clean record from **D**, the glitch score of the resulting data set is $g(\mathbf{D}^3) = 0.356976 > g(\mathbf{D})$.
The aggregate glitch score reflect the health of the data set and enables us to measure the effectiveness of a cleaning strategy.

IV. Discovering Patterns of Glitches

A. Estimating Glitch Distributions

Understanding the distribution of glitches, the interaction among glitches, and, dependence between glitches and the data set values, enables us to identify patterns that could help us design smart cleaning strategies. We estimate the distribution of glitches empirically. Let nk be the number of cells in **D**. Consider the *indicator function* I_{ij} where $I_{ij} = 1$, if d_{ij} is *glitchy* and has at least one non-null element in its signature ($m_{ij} > 0$), 0 otherwise. Let $N = \sum_{i,j} I_{ij}$ be the total number of cells that are categorized as glitches. The *maximum likelihood estimate* of the proportion of glitches at the value level in a given data set **D** is given by

$$\hat{P}_{\mathcal{G}} = \frac{\sum_{i,j} I_{ij}}{nk} = \frac{N}{nk}. \tag{5}$$

As n becomes large, $\hat{P}_{\mathcal{G}}$ converges to the true probability of findin a glitch in **D** (See Annexe B for details). Analogously, let $I_{ij}(c)$ be an indicator function such that $I_{ij}(c) = 1$ if the element d_{ij} has a glitch of category c (*i.e.*, $m_{ij}(c) \geq 1$). Then $N_c = \sum_{i,j} I_{ij}(c)$ is the number of glitches of category c in **D**. The empirical estimate of the *marginal distribution* of glitch category c, is given by

$$\hat{P}_c = \frac{\sum_{i,j} I_{ij}(c)}{nk} = \frac{N_c}{nk}. \tag{6}$$

B. Testing Independence of Glitch Types

Based on these estimates, we test the independence of different categories of glitches using *Pearson's Chi-square* test for independence. We test whether the signature data validates *the null hypothesis* (*i.e.*, that the glitch categories are independent). If two glitch categories are independent, the probability of the co-occurrence of the two categories c_1 and c_2 is obtained by multiplying the individual marginal probabilities as:

$$\hat{P}_{c_1,c_2} = \hat{p}_{c_1}\hat{p}_{c_2} = \frac{N_{c_1}}{nk}\frac{N_{c_2}}{nk} \tag{7}$$

We obtain the *expected number of co-occurrences under the null hypothesis,* E_{c_1,c_2} to be:

$$E_{c_1,c_2} = nk\hat{P}_{c_1,c_2} = \frac{N_{c_1}N_{c_2}}{nk}. \tag{8}$$

The actual *observed number of co-occurrences,* O_{c_1,c_2} in the data **D**, is given by $N_{c_1,c_2} = \sum_{i,j} I_{ij}(c_1, c_2)$. The disparity between the observed and expected is measured by Pearson's Chi-square test given by:

$$\chi^2(c_1, c_2) = \frac{(O_{c_1,c_2} - E_{c_1,c_2})^2}{E_{c_1,c_2}}. \tag{9}$$

The disparity is deemed to be statistically significan if it exceeds the known value of the standard Chi-square distribution at the desired level of significanc (see Annexes B and C for an explanation of hypothesis testing methodology. Annexe C also contains tests for extending the methodology to test the independence of q ($q \geq 2$) categories of glitches using a *multinomial test* for independence of q categories).

C. Discovery of Glitch Patterns

Pattern of glitches. Consider the glitch signature matrix $\mathbf{G} = \{g_{ij}\}$ computed from the observed data set **D**. Given a pair of threshold parameters (σ, ξ), a *glitch pattern* $\mathcal{P}(\sigma, \xi)$, is define to be a set of signatures {**g**} from **G**, if there exist *i)* at least σ distinct rows of **D** such that the values d_{ij} in these rows share the same signature **g** $\forall i$ and at least ξ values d_{ij} in each row with signature **g** $\forall j$, *ii)* σ and ξ are maximal, and *iii)* the detected glitch categories in **g** are strongly dependent (based on Eqs. 9 and 16). The thresholds (σ, ξ) ensure that the pattern is non-trivially prevalent across rows and attributes. In the rest of the section, we provide the details of the algorithm for discovering patterns of glitches . The algorithm for glitch pattern discovery (GPD) consists of two main steps: the construction of a Frequent Pattern tree (FP-tree) as proposed in [7] based on the glitch signature matrix and the actual mining for this data structure by building the tree of patterns of dependent glitch categories.

The goal of the GPD algorithm is firs to build the compact data structure called FGP-tree (*Frequent Glitch Pattern Tree*), which is a prefi tree in the glitch signature data set representing sub-signatures pertaining to a given minimum *vertical support* threshold (σ) corresponding to the minimum number of rows and a minimum *horizontal support* threshold (ξ) corresponding to a minimum number of values supporting the glitch pattern. The tree structure, called FGP-tree is a variation of the original FP-tree proposed by Han et al. [7]. The construction of the FGP-tree is done in two phases, where each phase requires a full I/O scan of the glitch signature matrix. A firs initial scan

of the database identifie q, the number of categories and the set of signatures with a non null response for a single category. It corresponds to the frequent 1-itemsets in FP-tree algorithm. After the enumeration of the non null responses appearing in the signatures, signatures with *vertical support* less than the threshold σ or *horizontal support* less than the threshold ξ are weeded out. Marginal and joint distributions (as define in Eqs. 6 and 15) are computed from signatures having more than two non null responses for distinct categories, and Chi-square test is computed for 2 to q-sets of categories. The signatures corresponding to independent categories, *i.e.*, with evidence of independence of glitch categories: if $\chi^2(C_1, \ldots, C_q) < \chi^2_\alpha$, are also removed from **G**. The remaining signatures are sorted by their frequency.

Algorithm 1 Glitch pattern discovery algorithm (GPD)

Input: Glitch matrix **G**; Support thresholds σ and ξ in [0,1]
Output: the set of Glitch Patterns with their respective supports and dependence values.
Scan **G** to fin the set of frequent non null responses for single glitch category $F1$
Scan **G** to build the Frequent Glitch Pattern Tree *FGP*
FGPB ← *FindFrequentGlitchPatternsBases*(*FGP*)
Patterns ← *FindMaximals*(*FGPB*, σ, ξ)
Output *Patterns*

Algorithm 1 shows the main steps in our approach. After building the FGP-tree, we mark some specifi nodes in the pattern lattice using *FindFrequentGlitchPatternBases*. Using the FGPBs, *FindMaximals* function discovers the maximal patterns at the frequent pattern border in the lattice. Algorithm

Algorithm 2 *FindFrequentGlitchPatternsBases*

Input: *FGP* (ReducedFP-tree based on chi-square test)
Output: *FGPB* (Frequent glitch patterns with counts)
ListNodesFlagged ← ∅
Follow the linked list of leaf nodes in *FGP*
for each leaf node *N* **do**
  Add *N* to *ListNodesFlagged* **end for**
while *ListNodesFlagged* ≠ ∅ **do**
  N ← Pop(*ListNodesFlagged*) {from top of the list}
  f ← Path from *N* to *root*
  f.branchSupport ← *N.supportV* - *N.counterV*
  f.branchSupport ← *N.supportH* - *N.counterH*
  for each node *P* in *f* **do**
    P.counterV ← *P.counterV* + *f*.branchSupport
    P.counterH ← *P.counterH* + *f*.branchSupport
    if *P.counterV* < *P.supportV*
      AND *P.counterH* < *P.supportH*
      AND ∀*c* child of *P*, *c.counterV* = *c.supportV*
        AND *c.counterH* = *c.supportH*
    then add *P* in *ListNodesFlagged* **end if**
  end for
  add *f* in *FGPB*
end while
Return *FGPB*

2 shows how patterns in the lattice are marked. The linked list of leaf nodes in the FGP-tree is traversed to fin upwards the unique paths representing sub-signatures. If frequent maximals exist, they have to be among these complete sub-signatures. Algorithm 3 traverses the lattice to fin maximals. It starts by listing some candidate maximals stored in *PotentialMaximals* which is initialized with the frequent pattern bases that are frequent. These FGPBs are stored in the list *List* and

Algorithm 3 *FindMaximals*

Input: FGPB (FrequentPattern Bases); σ, ξ (Support thresholds)
Output: Maximals (Frequent Maximal glitch patterns).
List ← *FGPB*; *PotentialMaximals* ← ∅
for each *i* in *List* **do**
  Find supports of *i* using branch supports
  if (*supportV*(*i*) > σ AND *supportH*(*i*) > ξ) **then**
    Add *i* to *PotentialMaximals* ;
    Remove *i* from *List* **end if**
end for
Sort *List* based on *supportV* ; *NList* ← *List*; *Nlist*2 ← ∅;
∀i∈ *NList* initialize *i*.fla ← *NULL* AND *i*.startpoint ← index of *i* in *NList*
while *NList* ≠ ∅ **do**
  for each *i* in *NList* **do**
    h ← *Intersect*(*i*, *j*) {where *j* ∈ *List* AND *i* << *j* (in lexicographic order)
    AND not *j*.fla }
    h.startpoint← *j*; Add *h* to *NList*2;
  end for
  for each *i* in *NList*2 **do**
    Find supports of *i* (using branch supports)
    if (*supportV*(*i*) > σ AND *supportH*(*i*) > ξ) **then**
      Add *i* to *PotentialMaximals*;
      Remove all duplicates or subsets of *i* in *NList*2;
      Remove *i* from *NList*2;
    else Remove all duplicates of *i* in *NList*2 except the most right one ;
      Remove *i* from *NList*2;
      Remove all non frequent subsets of *i* from *NList*2;
      if ∃*j* ∈ *NList*2 AND *j* ⊇ *i*
      then *i*.fla ← *j* **end if**
    for all *j* in *List* **do**
      if *j* >> *i*.startpoint (in lexicographic order) **then** *n* ← *Intersect*(*i*, *j*)
      Find supports of *n* (using branch supports)
        if (*supportV*(*n*) > σ AND *supportH*(*n*) > ξ)
          then Remove *i* from *NList*2 **end if end if**
    end for
    end if
  end for
  NList ← *NList*2; *NList*2 ← ∅;
end while
Remove any *x* from *PotentialMaximals* if (∃*M* ∈ *PotentialMaximals* AND *x* ⊂ *M*)
Maximals ← *PotentialMaximals*
Return *Maximals*

intermediate lists *NList* and *NList*2 will store the nodes in the lattice that the intersection of FGPBs would point to, in other words, the nodes that may lead to maximals. The nodes in the lists have two attributes: *flag* and *startpoint*. For a node *n*, fla indicates that a subtree in the intersection tree should not be considered starting from the node *n*. For example, if node (D ∧ O) has a fla *M*, then the subtree under the node (D ∧ M ∧ O) should not be considered (See Fig.2). For a given node *n*, *startpoint* indicates which subtrees in the intersection tree, descendants of *n*, should be considered. For example, if a node (D ∧ I) has the startpoint *O*, then only the descendants (D ∧ I ∧ O) and so on are considered, but (D ∧ M ∧ O) is omitted. Note that glitch categories are ordered lexicographically (DIMO – Duplicate, Inconsistent, Missing, Outlier). At each level in the intersection tree, when *NList*2 is updated with new nodes, the support thresholds are used to prune the intersection tree. The same process is repeated for all levels of the intersection tree until there are no other intersections to do (*i.e.*, *NList*2 is empty). At the end, the fina set is cleaned by removing subsets of any sets in *PotentialMaximals*.

Example 5: Fig. 2 shows the patterns discovered in the **SNMP** data set, one of the three real-world data sets used in our experiments (see Section VI). The rows and boxes

highlighted in green correspond to statistically significan departures from independence between pairs and triplets of D=Duplicate, I=Inconsistent, M=Missing and O=Outlier categories, as measured by high Chi-square and low p-values ($p = .05$). In the SNMP data set, the strongest patterns of dependence are dependencies occurring as pairs of concomitant outlying or inconsistent values within duplicate records for the same rows (*i.e.*, DI, DO, IO) and as triplets of missing or inconsistent values within duplicate records (*i.e.*, DIM and DIO) as shown in Fig. 2.

Based on these discovered patterns and on Table II given in Annexe D, we selected cleaning strategies involving either method #4 removing the complete rows with concomitant glitches or any of the replacement methods (from #5 to #14 of Table D) applied to the rows with the discovered patterns of concomitant glitches.

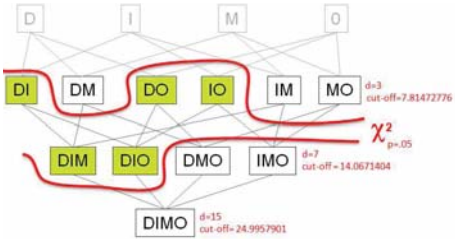


Fig. 2. Glitch patterns discovered from SNMP data set

Annexe E presents patterns of glitches discovered in two other real-world data sets used in our experiments. Discovering patterns of glitches in the real-world data sets enabled us to propose cleaning strategies that simultaneously eliminated multiple co-occurring glitches, significantl reducing the number of cleaning steps, thereby reducing the overall cost.

V. Quantitative Cleaning

A. Computing Effectiveness and Cost

A *cleaning strategy* is a sequence of methods applied to transform an input data set to clean the instances of one or more glitch categories.

The effectiveness of a cleaning strategy S with respect to a data set $\mathbf{D}$ denoted by *eff*(S) is measured by the proportion of change in the quality of $\mathbf{D}$. If a strategy S is composed of M cleaning methods applied sequentially to the input data set D^0, then:

$$eff(S) = \frac{g(D^0) - g(D^S)}{g(D^0)} = \sum_{m=1}^{M} \frac{g(D^{m-1}) - g(D^m)}{g(D^{m-1})}.$$

Normalized effectiveness is given by

$$Eff(S) = \frac{eff(S)}{\sqrt{eff(S)^2 + a}} \tag{10}$$

where $a \geq 0$ is a normalization constant. $Eff(S) \in [-1; +1]$. If $Eff(S) = 0$, then the strategy S does not change the glitch score of the input data set. $Eff(S)$ tends to -1 when the strategy degrades the data quality and increases the glitch score, and conversely, $Eff(S)$ tends to $+1$ when the strategy improves the quality of data as reflecte in a reduction in the initial glitch score. The cost of a cleaning strategy S is the sum of the costs of the individual methods:

$$Cost(S) = \sum_{m=1}^{M} \Omega_m \frac{g(D)}{size(D)} \tag{11}$$

The cost of each method m is estimated as a function of the size of the input data set D, its glitch score, and Ω_m, the average unit time taken by the method m to correct one glitch entity. Annexe D provides average cost and effectiveness of the methods we used in our experiments.

B. Generating the Set of Candidate Strategies

Since the space of potential cleaning strategies can be quite large, we have to reduce it by specifying constraints. As the fina step of the DEC framework, we propose the following strategy elimination methods:

1) *Cost based*: Strategies whose cost exceeds an upper bound **U** are eliminated. For example, imputing missing values using point estimates such as the median would be retained, but an expensive simulation-based method that estimates the joint distribution will be eliminated.

2) *Priority based*: Certain glitches are high priority (e.g., "glitch score exceeds the effectiveness threshold $\mathbf{\Gamma}$") and have to be cleaned. The strategies that eliminate the set of "must-treat" glitches $\mathbf{T} \subseteq \mathbf{D}$ is given by $\mathbf{T} = \{d_{ij} | g_{ij} \geq \mathbf{\Gamma} > 0\}$. We propose three ways to choose the effectiveness threshold $\mathbf{\Gamma}$:

- *Resource driven* (DEC-RD). If the user has the resources to treat just the top k or top p% of glitches, then $\mathbf{\Gamma}$ is the k^{th} ranked glitch score or the p^{th} percentile of the glitch scores respectively.
- *Specificatio driven* (DEC-SD). If the user specifie that a certain glitch category t must be treated, the weight function should reflec this in the glitch score to make such glitches more influential
- *Model driven* (DEC-MD). When there are no pre-requisites regarding the glitch treatment, the user can choose a cut-off based on an expected improvement of the aggregate glitch score: $Eff(S) \geq \mathbf{\Gamma} \geq 0$.

3) *Pattern based* (PD): In addition, we can take advantage of glitch patterns to improve the cleaning strategy by applying the same treatment to glitches that are significantl correlated. All the previous approaches can be then extended to take into account the discovered patterns.

Finally, for any data set $\mathbf{D}$, we generate the set of candidate strategies $\mathcal{S}$ with a total order on $\mathcal{S}$ such that *i)* strategies are ordered by non-decreasing cost and *ii)* the search chooses the firs strategy in the ordering that has the best effectiveness.

C. Finding the Proximal Cleaning Strategies

Ideal data set. As discussed in Section II-A, in order to fin proximal strategies, the user has to defin an ideal data set $\mathbf{D}^\star$. The ideal data set can be specifie in many ways, e.g. the number and categories of glitches the user is willing to tolerate, or a mathematical model that the cleaned data set should

conform to, or a set of constraints that $\mathbf{D}^\star$ should satisfy. The ideal data set could potentially contain an expected but acceptable number of glitches. The list of candidate cleaning strategies $\mathcal{S}$ is composed of the strategies that are selected based on cost and effectiveness constraints and ordered by non-decreasing cost. From among the resulting data sets, we choose a cleaning strategy B such that the resulting data set D^B is the closest to the ideal data set $\mathbf{D}^\star$:

$$D^B = \arg\min_{\{S \in \mathcal{S}\}} [\mathtt{dist}(D^S, \mathbf{D}^\star)]$$
$$\text{subject to } Cost(S) \leq \mathbf{U} \text{ and } Eff(S) \geq \mathbf{\Gamma} > 0 \qquad (12)$$

$Cost(S)$ is the cost of strategy S, $\mathbf{U}$ is a pre-define upper bound for the cost; $\mathbf{\Gamma}$ is the lower bound for effectiveness $Eff(S)$ of the cleaning strategy S. `dist` is a distance function that measures the distance between D^S, the result of strategy S, and $\mathbf{D}^\star$ the ideal data set. Ultimately, if two strategies result in clean data sets that are equidistant from $\mathbf{D}^\star$, the data set that is the closest to the original data set $\mathbf{D}$ will be selected. Examples of statistical distances range from the computationally simple, like the multinomial distance (measures the difference in the distribution of the glitch categories) to the complex, like the Kullback-Leibler distance between the actual data distributions of D^S and $\mathbf{D}^\star$ (See Annexe C for details).

VI. Experiments

In this section, we present experiments to demonstrate that the DEC approach significantl outperforms the conventional quantitative cleaning strategies in terms of accuracy, closeness to the ideal dataset, and scalability. We implemented our methods in SAS 9.2 and Java 1.6, and conducted experiments on a Windows XP machine with AMD Athlon(tm) 64 2GHz CPU and 960MB memory.

Datasets. We use three real-world data sets: (1) Patent data from EPO[1] with missing and incomplete string values, duplicates, and inconsistent addresses. (2) Sensor data collected in the Intel Berkeley Lab[2], riddled with missing, truncated, and inconsistent values, detected based on range constraints and violations of attribute correlations. (3) An internal SNMP data set[3], containing outliers, missing values, and duplicate records. To test the scalability of our method, we generate two sets of very large data glitch signature data sets of varying sizes (from 100 M to 504 M binary responses). The firs set is a control version that has no patterns of glitches. In the second one, we inject 3 types of patterns involving 2, 3, and 4 glitch categories and vary the proportions of concomitant, multi-type and multi-occurrent glitches. We also use the sensor data set for scalability evaluation. We report the computation time for proximal strategy selection and execution. We defin three types of *ideal* data sets. `Ideal1` excludes all imperfect data, `Ideal2` contains a total of g% of glitches drawn Uniformly or Randomly from the glitch categories, with g varying from 5 to 30% of the data set size, and `Ideal3` preserves either the original Median or an attribute relationship specifie by a Regression model.

Detection. We focus on four categories of glitches: duplicate records (`D`), inconsistent (`I`), missing (`M`), and outliers (`O`). Duplicates are detected with 10 distance-based record linkage methods from Java SimMetrics library[4]. We check null values and a small set of constraints (based on ranges and known correlations) specifi to each dataset. Outliers are identifie with 10 SAS procedures[5]. We compute the glitch signatures and scores for each data set.

Exploration. For each data set we use the glitch signature set to test independence between glitch categories as define in Section IV, and identify categories that can be treated independently. We use the GPD algorithm given in Section IV-C to discover glitch patterns.

Cleaning. We describe the quantitative cleaning methods we use,

along with their average cost and effectiveness in Annexe D. We identify the candidate set of strategies based on cost and effectiveness thresholds, assigned priority and discovered patterns. Finally, we compute the distance between the clean data sets created by the candidate strategies and the ideal datasets, and select a proximal cleaning strategy as define in Section V-C Eq. 12.

Focus of the study. We study the effect of the following parameters on cleaning strategies (1) g: the percentage of glitches injected, (2) M_c: the number of detection methods per category, and (3) n: the data set size (in tuples).

Evaluation of the approach. We evaluate our approach using precision and recall measures and we report F-measure define as $F\text{-}measure = \frac{2 \times Precision \times Recall}{Precision + Recall}$.

Detection F-measure. The detection precision of the DEC process is the ratio of the number of correctly detected glitches to the total number of glitches detected by a method. Recall is given by the ratio of the number of correctly detected glitches to the total number of glitches in the data set. For missing and inconsistent values, the computation of detection F-measure is straightforward. For outliers, we fla anomalous values to establish the ground truth and synthetically include new anomalies by changing "normal" values of a randomly selected attribute. For duplicates, we inject approximate duplicate records. We repeat each pollution scenario 10 times. Since we know the normal values/records, we compute the detection F-measure for each polluted version of each real-world data

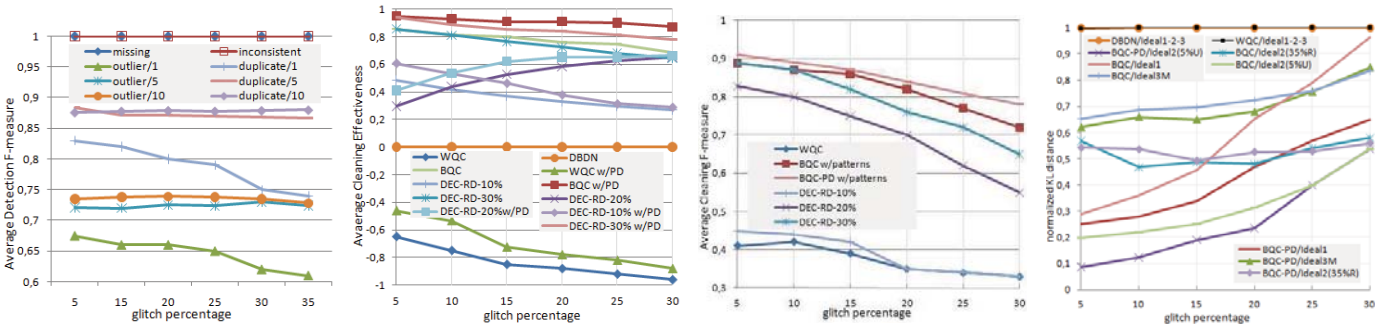
[1]Data set of inventors with 754,075 records, 4 non-key attributes (string, categorical and numerical data) from EPO (European Patent Office): `http://www.epo.org/patents/patent-information/raw-data.html`.

[2]Data set of 2,313,682 million readings, 8 attributes (timestamp, sensorID, temperature, light, voltage) collected every 31 seconds from 54 sensors deployed in the Intel Berkeley Research lab between February 28th and April 5th, 2004: `http://db.csail.mit.edu/labdata/labdata.html`.

[3]SNMP data set (8,632 tuples, 11 variables) collected every 5 minutes during one month (timestamps, categorical and numerical values).

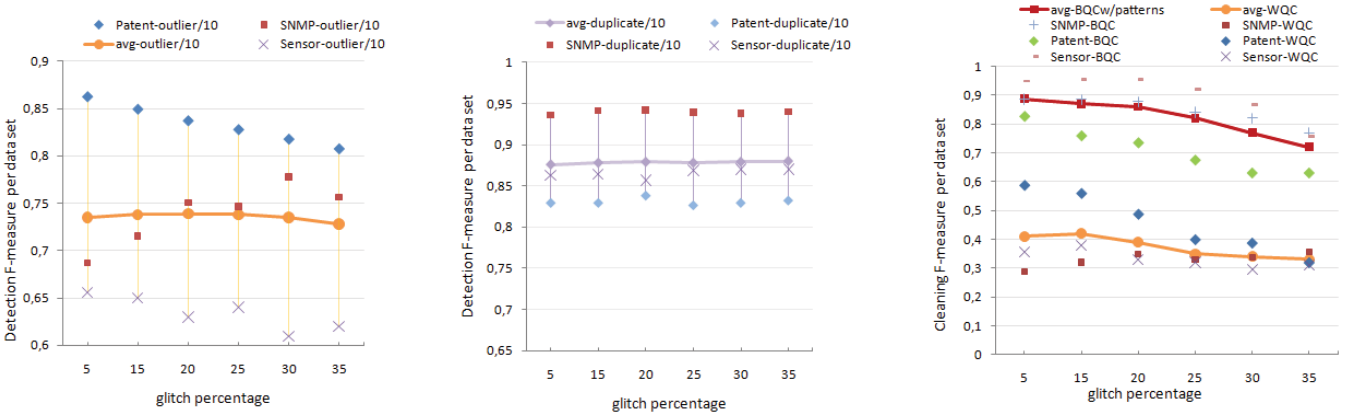
[4]Namely, Levenshtein, Edit, Jaro, Jaro-Winkler, SoundEx, Monge-Elkan, Hamming, cosine, q-gram, and Jaccard distances from SimMetrics library: `http://sourceforgenet/projects/simmetrics/`.

[5]SAS 9.2: `http://support.sas.com/documentation/.`, see documentation related to `UNIVARIATE`, `RCHART`, `BOXPLOT LOW/HIGH`, `FARLOW/FARHIGH`, `MVE`, `MCD`, `ROBUSTREG` SAS procedures



(a) Average detection F-measures **(b)** Average cleaning effectiveness **(c)** Average cleaning F-measure **(d)** Distance to Ideal data sets

Fig. 3. **Accuracy evaluation experiments.** Pattern discovery always improves the performance when glitch percentage increases.



(a) Outlier detection F-measures **(b)** Duplicate detection F-measures **(c)** Cleaning F-measures

Fig. 4. F-measures for the real-world data sets

set, test several sets of detection methods to study κ, the degree of agreement in detection with 2 to 10 detection methods, and report the average detection F-measure and variance per data set.

Cleaning F-measure. We use the pre-define ideal data sets: `Ideal1` and 10 different configuration for `Ideal2` and `Ideal3` to benchmark the accuracy of our cleaning strategies, and report the average measures and variance per data set. Precision of a cleaning strategy is the ratio of the number of glitches that have been cleaned to exactly match the ideal data set, to the total number of glitches cleaned by the strategy. The recall of a cleaning strategy is the ratio of the number of glitches cleaned to exactly match the ideal data set, to the total number of glitches. We report the evaluations of the following strategies:

- `DBDN`: Detect But Do-Nothing,
- `DEC`: the following variants of the DEC approach:
  - `DEC-RD` with resource-driven selection of the best strategies;
  - `DEC-SD` with specification-dr ven selection,
  - `DEC-MD` with model-driven selection,
- `BQC`: the best quantitative cleaning strategy,
- `WQC`: the worst cleaning strategy from the candidate set.

We use the suffix `-PD` for the strategies based on pattern discovery.

A. Accuracy Evaluation

Detection F-measure. Fig. 3(a) compares the accuracy of a single detection method against a combination of 2 to 10 methods. The *Y*-axis shows average detection F-measure plotted against the percentage of glitches g on the *X*-axis. As the percentage of glitches increases, the F-measure decreases in almost all cases. However, the F-measures are relatively stable when 5 and 10 methods are used for outlier and duplicate detection. Relying on a single method shows the worst performances for both detections (`outlier`/1 and `duplicate`/1). Figures 4(a) and (b) show the F-measures for each real-world data set for outlier and duplicate detection. Outlier methods have greater variability in detection F-measure relative to duplicate detection methods due to different degrees of sensitivity and robustness. This experiment confirm the need for using multiple detection methods for better detection performance.

Cleaning Effectiveness. Fig. 3(b) shows average effectiveness measure for DEC strategies composed of the methods given in Annexe D. We plot the effectiveness of the best and worst quantitative cleaning strategies `BQC` and `WQC`, and cleaning effectiveness of DEC methods subject to resource constraints (`RD`) in treating top 10, 20, and 30% glitches, with and without glitch pattern discovery. We injected three types of patterns using the four glitch categories with various proportions of concomitant, multi-type and multi-occurrent

glitches. We replicate the experiments 10 times using multiple combinations of dependence between the 2, 3 and 4 glitch categories for each data set. We report the average cleaning effectiveness over the replications. DBDN strategy has null effectiveness. DEC-RD-10% and 20% F-measures decrease as the percentage of glitches increases; DEC-RD-30% performs better than the other RD strategies (without pattern discovery) for data sets with glitch percentage under 15%, but not when more glitches are introduced mainly because of pre-existing glitches in the real-world data sets. Strategies that use pattern discovery (*-PD) maintain a superior effectiveness value (by +8% on average) compared to the original strategies that do not use PD. However, the methods do not achieve an effectiveness of 1 mainly because they may introduce new glitches (in particular, when imputation methods are inappropriate).

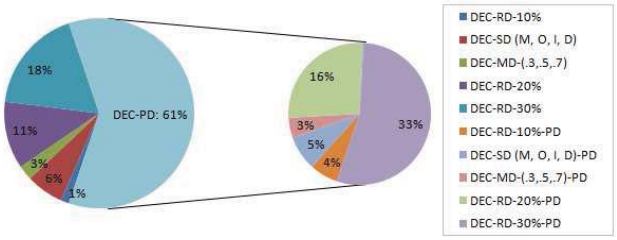
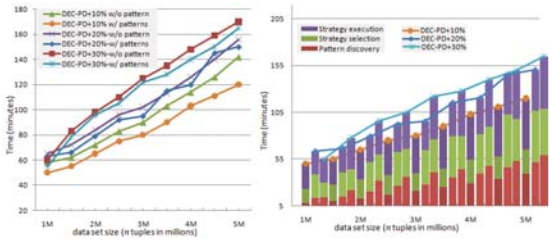


Fig. 5. Distribution of BQC method types

Fig. 5 shows the distribution by type of the best quantitative cleaning strategies in the set of 1000 experiments for the proximal strategy selection (using 10 different configuration for the ideal data sets and 10 versions of real-world data sets polluted with glitches injection). The -PD methods dominate the best strategies. DEC-PD constitutes 61% of the BQC strategies with Resource-Driven methods for the top 30% of glitches (DEC-RD-30%-PD) contributing 33%.

Cleaning F-measure. Fig. 3(c) shows the average cleaning F-measures for each strategy, averaged over the F-measures of 10 versions of each real-world data sets with respect to 10 different configuration of the ideal data sets, plotted against the percentage of injected glitches. BQC performs slightly better with pattern discovery (PD) than without. Cleaning F-measures computed with respect to Ideal2 with 20 to 30% glitches randomly injected are the worst, pulling down the average F-measure in Fig. 3(c). Fig. 4(c) shows the variability of cleaning accuracy in the real-world data sets.

Distance to Ideal data sets. Fig. 3(d) shows the normalized KL distance of the result of the best cleaning strategies with and without pattern discovery. The more the glitches (with no pattern), the farther the cleaned data are from ideal data sets, and less the gain offered by pattern discovery over the original cleaning strategies. Fig. 3(d) also shows that the distance to the ideal data sets is a complementary and more precise measure than the cleaning F-measure to evaluate the quality of the cleaning strategies. A low F-measure in Fig. 3(c) may correspond to an acceptable distance to an ideal data set. Conversely, WQC, the farthest strategies along with DBDN, has a cleaning F-measure value greater than 0.33, but the cleaned data set corresponding to DBDN is meaningless in the



(a) With and without glitch pattern **(b)** decomposed time for PD methods

Fig. 6. DEC methods scalability

context of the user-define ideal data set. This is clear evidence that exploiting the patterns of glitches results in efficient and cost effective strategies that are far superior to existing blind quantitative cleaning strategies.

B. Scalability Evaluation

In this section, we investigate the scalability of DEC methods. We use *(i)* the sensor data set, which contains about 2.3 M tuples (18.4 M values) with a glitch signature matrix of 405 M binary responses[6] and *(ii)* a synthetic data set with a generated glitch signature matrix of 900 M binary responses (30 M values × 30 detection methods). Fig. 6(a) shows the scalability of the DEC methods. We plot the execution time against the data/glitch set size, for DEC methods with (w/patterns) and without (w/o pattern) pattern discovery, for injected glitch levels of 10, 20 and 30%. Our approach scales well for large data sets. The PD methods usually require more time, but perform slightly better than the original methods. The patterns of glitches are used to optimize the cleaning operations by reducing the number of individual glitches that need to be treated. All the methods degrade with increased number of glitches. DEC-PD is generally more efficient when patterns exist, because treating a single pattern could clean multiple glitches, values, and tuples. The difference in execution time between DEC-PD w/ and DEC-PD w/o patterns decreases in proportion to the increase in glitches. In particular, Fig. 6(b) shows the respective execution times for each step of DEC-PD methods, *i.e.*, pattern discovery using GPD algorithm, strategy selection, and execution when data set size increases for the three levels of injected glitch percentage: 10%, 20%, and 30%.

Fig. 7 shows execution time for GPD by glitch type and proportion such as mixture of concomitant cc, multi-occurrent mo and multi-type mt glitches when 20% of glitches are injected.

The minimum execution time corresponds to the case where 100% of multi-type glitches (100%-mt) are injected which is predictable because several glitch categories can be cleaned simultaneously. The maximal execution time always corresponds to the case where 100% of multi-occurrent glitches (100%-mo) are injected mainly because of the cost of Chi-square test computation and pattern discovery.

[6] 2.3 M tuples × 8 attributes × 22 detection methods.

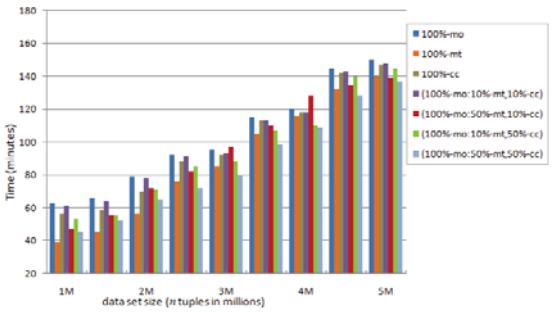


Fig. 7. DEC-PD-20% execution time with varying proportions of multi-occurrent, multi-type, and concomitant glitches in the injected patterns

VII. Related Work

There has been much work in the database research [8], data mining and statistics communities toward developing and adapting data mining-style techniques for the detection and treatment of data glitches [9]. The work can be classifie into two disjoint approaches: 1) diagnostic approaches derived from quantitative methods such as robust estimators, methods for univariate and multivariate outlier detection [12], methods for distributional change detection [4] and 2) corrective approaches such as procedural data transformation via ETL operators [3], record linkage techniques [5], and quantitative data cleaning [8] such as missing value imputation [17], [14]. In addition, there has been considerable recent research in the area of constraint-based data repair, where conditional FDs (CFDs) [2] are exploited to audit and repair the data [15], [16], [13]. However, most of the techniques detect or treat each data glitch in isolation, and they do not exploit the glitch patterns for data cleaning. The detection is also clearly independent and disconnected from the cleaning process.

The main challenges of these techniques is the size of the data set and the complex relationships between glitches that co-occur or recur with a non-random structure when they have a common root cause in the system or process that generates them. Our approach addresses the challenges eluded by prior work.

VIII. Conclusion and Perspectives

In this paper, we proposed DEC, an iterative framework for detecting, exploring and cleaning complex data glitches. We defin different types of complex glitches, and develop heuristics and data driven strategies for quantitative cleaning using the patterns and joint distributions of these glitches. In doing so, we address several short comings in extant approaches, by linking detection and cleaning through iteration and expert feedback, and by treating inter-related glitches simultaneously rather than one at a time. These strategies are more effective than traditional strategies. We provide a rigorous statistical basis for selecting an optimal cleaning strategy from a candidate set, and demonstrate the effectiveness, accuracy, and scalability of our approach. Data quality mining in general, and treatment of complex glitches and their patterns in particular, is a new and rich fiel of research. Our future work will focus on novel methods for pattern discovery, glitch detection, classification and scoring; innovative optimization algorithms for strategy selection. In addition, we will combine our approach with CFD-based audit and repair approaches and consider scenarios where just aggregate indicators of the glitch distribution are available.

References

[1] L. Berti-Équille and T. Dasu. New directions in data quality mining. Tutorial KDD 2009.
[2] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for data cleaning. In *ICDE 2007*.
[3] T. Dasu and T. Johnson. *Exploratory Data Mining and Data Cleaning*. 2003.
[4] T. Dasu, Krishnan S., D. Lin, S. Venkatasubramanian, and K. Yi. Change (detection) you can believe in findin distributional shifts in data streams. In *IDA*, 2009.
[5] A. Elmagarmid, P. Ipeirotis, and V. Verykios. Duplicate record detection a survey. *IEEE TKDE*, 19, 2007.
[6] J.L. Fleiss. *Statistical methods for rates and proportions*. 1981.
[7] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *SIGMOD 2000*.
[8] J. Hellerstein. Quantitative data cleaning for large databases. Technical report, UC Berkeley, Feb. 2008.
[9] J. Hipp, U. Güntzer, and U. Grimmer. Data quality mining - making a virtue of necessity. In *DMKD*, 2001.
[10] M. Hua and J. Pei. Cleaning disguised missing data: a heuristic approach. In *KDD 2007*.
[11] R.B. Kline. *Data Preparation and Screening in Principles and Practice of Structural Equation Modeling*. 2005.
[12] H.-P. Kriegel, P. Kroger, and A. Zimek. Outlier detection techniques. Tutorial PAKDD 2009.
[13] C. Mayfield J. Neville, and S.Prabhakar. ERACER: a database approach for statistical inference and data cleaning. In *SIGMOD 2010*.
[14] J. L. Schafer. *Analysis of Incomplete Multivariate Data*. 1997.
[15] D. Srivastava. Data auditor: Analyzing data quality using pattern tableaux. In *ER 2009*.
[16] M. Yakout, A. Elmagarmid, J. Neville, and M. Ouzzani. GDR: A System for Guided Data Repair. In *SIGMOD 2010*.
[17] Y. C. Yang. Multiple imputation for missing data: Concepts and new development. *SAS Institute, Inc.*, 2000.

Appendix

Annexe A. Corroborating Multiple Detection Methods with Fleiss' Kappa Measure

Fleiss' Kappa measure for the glitch category c is define as

$$\kappa_c = \frac{A_c - E_c}{1 - E_c}. \tag{13}$$

For methods that detect the glitch category c, the factor $A_c - E_c$ gives the degree of agreement attainable above chance, and $1 - E_c$ gives the degree of agreement actually achieved. These factors are define as $E_c = \frac{1}{nkM_c} \sum_{i,j,l} \left(a_{ijl}(c)\right)^2$ where $a_{ijl}(c)$ is the number of methods which detected value d_{ij} as a glitch of category c ($l = 1$) or clean ($l = 0$). M_c is the total number of methods used for detecting the glitch category c. A_c is the number of pairs of agreement between the methods returning the same detection response. The total proportion of agreement of the methods is obtained from the set of observations as $A_c = \frac{1}{nkM_c(M_c - 1)} \left(\sum_{i,j,l} \left(a_{ijl}(c)\right)^2 - M_c \right)$.

Annexe B. Estimating Glitch Distributions

The empirical estimate of the *conditional probability* of the occurrence of a glitch that belongs to category c given that there are N glitches, is given by:

$$\hat{P}_{c|\mathcal{G}} = \frac{N_c}{N}. \tag{14}$$

Consider the empirical estimate of the joint distribution of two categories of glitches c_1 and c_2. Let $I_{ij}(c_1, c_2) = I_{ij}(c_1).I_{ij}(c_2)$. Then $N_{c_1,c_2} = \sum_{i,j} I_{ij}(c_1, c_2)$, the total number of cells, *i.e.*, d_{ij}s that have glitches of category c_1 and c_2, irrespective of the presence or absence of other glitch categories. Then, the joint distribution of glitch categories c_1 and c_2 is given by:

$$\hat{P}_{c_1,c_2} = \frac{\sum_{i,j} I_{ij}(c_1, c_2)}{nk} = \frac{N_{c_1,c_2}}{nk}. \tag{15}$$

This definitio can be extended to more than two categories in exactly the same way, by counting the number of cells that have the simultaneous occurrence of the given category of glitches. Note that all the estimates can be proved to converge asymptotically to the true probabilities by applying the Central Limit Theorem to the averages of the indicator functions.

Annexe C. Statistical Distances

An example of a simple statistical distance is the *multinomial distance* which computes the difference in the distribution of different categories of glitches between the transformed and the ideal data set based on Pearson's statistic. This is easy to compute since it requires just the counts of different categories of glitches. Let the number of categories of glitches in $\mathbf{D}^S$ be $(N_1, N_2, \ldots, N_q)$ and in $\mathbf{D}^\star$ be $(N_1^\star, N_2^\star, \ldots, N_q^\star)$. Under the null hypothesis that there is no significan difference in the two glitch distributions, we can pool the two vectors and get the expected proportion in each glitch category c to be $P_c = \frac{\left(N_c + N_c^\star\right)}{\sum_c \left(N_c + N_c^\star\right)}$. We then construct the test statistic

$$\chi^2 = \sum_c \frac{\left(N_c - P_c \sum_c N_c\right)^2}{P_c \sum_c N_c} + \sum_c \frac{\left(N_c^\star - P_c \sum_c N_c^\star\right)^2}{P_c \sum_c N_c^\star} \tag{16}$$

where χ^2 has a Chi-square distribution with $C-1$ degrees of freedom. An alternative is the *Kullback Leibler* distance which uses the concept of relative entropy to compute the distance between the *actual data distributions* of the data sets D and $\mathbf{D}^\star$

$$\texttt{dist}_{KL}(D, \mathbf{D}^\star) = \sum P_D \log \frac{P_D}{P_{\mathbf{D}^\star}}. \tag{17}$$

The distance has an asymptotically Chi-square distribution. This is a more complex test since we need to estimate the two actual multivariate data distributions, an expensive task. There are several methods of computing these complex distributions and their distances (See [4] for more details).

Annexe D. Quantitative Cleaning Methods

Table D presents the set of quantitative cleaning methods we used in our evaluation experiments. The normalized average cost and effectiveness values of each method (as define in Eq. 10 and 11) have been computed for the three real-world data sets (See details in SAS Documentation[7]) for imputation procedures.

Table II. AVAILABLE QUANTITATIVE CLEANING METHODS

#	Description	AvgCost	AvgEff
1	Doing nothing: The value remains an untreated glitch	0	-1
2	New category: Treat glitches as a separate category	0.3256	0.78563
3	List-wise deletion: Reject columns with glitches	0.1523	-0.4524
4	Pair-wise deletion: Reject rows with a glitch	0.1235	-0.3856
5	Single imputation based on the median	0.4587	0.3451
6	Imputation based on the cluster mean	0.6897	0.4812
7	Imputation based on regression	0.8623	0.4526
8	Multiple imputation based on EM	0.8253	0.4917
9	Multiple imputation based on regression	0.7988	0.2679
10	Multiple imputation based on predictive mean matching	0.7113	0.6891
11	Multiple imputation based on propensity score	0.8411	0.5677
12	Multiple imputation based on logistic regression	0.8394	0.7852
13	Multiple imputation based on discriminant function	0.8231	0.6614
14	Multiple imputation based on MCMC	0.8945	0.4219

Annexe E. Discovered patterns in the real-world data sets

Figures F1 and F2 present in green color the glitch patterns discovered respectively in Sensor and Patent data sets with their respective Chi-square values and p-values.

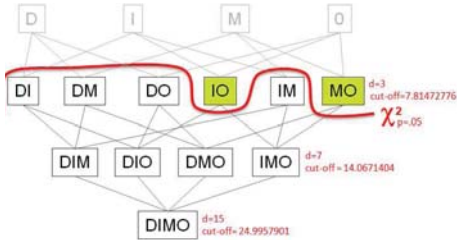


Figure F1. Glitch patterns discovered in Sensor data set

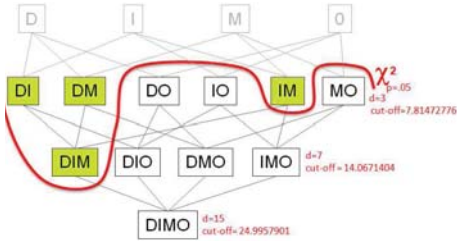


Figure F2. Glitch patterns discovered in Patent data set

ACKNOWLEDGEMENT

Laure Berti-Équille's mobility research program was supported by the European Commission (Grant FP6-MOIF-CT-2006-041000).

[7] http://support.sas.com/documentation/.