Interaction between record matching and data repairing

Citation for published version:
Fan, W, Li, J, Ma, S, Tang, N & Yu, W 2011, Interaction between record matching and data repairing. in Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011. ACM, pp. 469-480. DOI: 10.1145/1989323.1989373

Digital Object Identifier (DOI):
10.1145/1989323.1989373

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Early version, also known as pre-print

Published In:
Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011

Interaction between Record Matching and Data Repairing

Wenfei Fan[1,2] Jianzhong Li[2] Shuai Ma[3] Nan Tang[1] Wenyuan Yu[1]
[1]University of Edinburgh [2]Harbin Institute of Technology [3]Beihang University
{wenfei@inf., ntang@inf., wenyuan.yu@}ed.ac.uk lijzh@hit.edu.cn mashuai@act.buaa.edu.cn

Abstract

Central to a data cleaning system are record matching and data repairing. Matching aims to identify tuples that refer to the same real-world object, and repairing is to make a database consistent by fixing errors in the data by using constraints. These are treated as separate processes in current data cleaning systems, based on heuristic solutions. This paper studies a new problem, namely, the interaction between record matching and data repairing. We show that repairing can effectively help us identify matches, and vice versa. To capture the interaction, we propose a uniform framework that seamlessly unifies repairing and matching operations, to clean a database based on integrity constraints, matching rules and master data. We give a full treatment of fundamental problems associated with data cleaning via matching and repairing, including the static analyses of constraints and rules taken together, and the complexity, termination and determinism analyses of data cleaning. We show that these problems are hard, ranging from NP- or coNP-complete, to PSPACE-complete. Nevertheless, we propose efficient algorithms to clean data via both matching and repairing. The algorithms find *deterministic fixes* and *reliable fixes* based on confidence and entropy analysis, respectively, which are more accurate than possible fixes generated by heuristics. We experimentally verify that our techniques significantly improve the accuracy of record matching and data repairing taken as separate processes, using real-life data.

Categories and Subject Descriptors

H.2 [**Database Management**]: General—*integrity*

General Terms

Theory, Algorithms, Experimentation

Keywords

conditional functional dependency, matching dependency, data cleaning

1. Introduction

It has long been recognized that data residing in a database is often dirty [32]. Dirty data inflicts a daunting cost: it costs US businesses 600 billion dollars each year [16]. With this comes the need for data cleaning systems. As an example, data cleaning tools deliver "an overall business value of more than 600 million GBP" each year at BT [31]. In light of this, the market for data cleaning systems is growing at 17% annually, which substantially outpaces the 7% average of other IT segments [22].

There are two central issues about data cleaning:

- *Recording matching* is to identify tuples that refer to the same real-world entity [17, 26].
- *Data repairing* is to find another database (a candidate repair) that is consistent and minimally differs from the original data, by fixing errors in the data [4, 21].

Most data cleaning systems in the market support record matching, and some also provide the functionality of data repairing. These systems treat matching and repairing as separate and independent processes. However, the two processes typically interact with each other: repairing helps us identify matches and vice versa, as illustrated below.

Example 1.1: Consider two databases D_m and D from a UK bank: D_m maintains customer information collected when credit cards are issued, and is treated as *clean master data* [28]; D consists of transaction records of credit cards, which may be dirty. The databases are specified by schemas:

card(FN, LN, St, city, AC, zip, tel, dob, gd),
tran(FN, LN, St, city, AC, post, phn, gd, item, when, where).

Here a card tuple specifies a UK credit card holder identified by first name (FN), last name (LN), address (street (St), city, zip code), area code (AC), phone (tel), date of birth (dob) and gender (gd). A tran tuple is a record of a purchased item paid by a credit card at place where and time when, by a UK customer who is identified by name (FN, LN), address (St, city, post code), AC, phone (phn) and gender (gd). Example instances of card and tran are shown in Figures 1(a) and 1(b), which are fractions of D_m and D, respectively (the cf rows in Fig. 1(b) will be discussed later).

Following [18, 19], we use conditional functional dependencies (CFDs [18]) φ_1–φ_4 to specify the consistency of tran data D, and a matching dependency (MD [19]) ψ as a rule for matching tuples across D and master card data D_m:

φ_1: tran([AC = 131] → [city = Edi]),
φ_2: tran([AC = 020] → [city = Ldn]),
φ_3: tran([city, phn] → [St, AC, post]),
φ_4: tran([FN = Bob] → [FN = Robert]),
ψ: tran[LN, city, St, post] = card[LN, city, St, zip] ∧ tran[FN] ≈ card[FN] → tran[FN, phn] ⇌ card[FN, tel],

where (1) the CFD φ_1 (resp. φ_2) asserts that if the area code is 131 (resp. 020), the city must be Edi (resp. Ldn); (2) CFD φ_3 is a traditional functional dependency (FD) asserting that city and phone number uniquely determine street, area code

	FN	LN	St	city	AC	zip	tel	dob	gd
s_1:	Mark	Smith	10 Oak St	Edi	131	EH8 9LE	3256778	10/10/1987	Male
s_2:	Robert	Brady	5 Wren St	Ldn	020	WC1H 9SE	3887644	12/08/1975	Male

(a) Master data D_m: An instance of schema card

	FN	LN	St	city	AC	post	phn	gd	item	when	where
t_1:	M.	Smith	10 Oak St	Ldn	131	EH8 9LE	9999999	Male	watch, 350 GBP	11am 28/08/2010	UK
cf	(0.9)	(1.0)	(0.9)	(0.5)	(0.9)	(0.9)	(0.0)	(0.8)	(1.0)	(1.0)	(1.0)
t_2:	Max	Smith	Po Box 25	Edi	131	EH8 9AB	3256778	Male	DVD, 800 INR	8pm 28/09/2010	India
cf	(0.7)	(1.0)	(0.5)	(0.9)	(0.7)	(0.6)	(0.8)	(0.8)	(1.0)	(1.0)	(1.0)
t_3:	Bob	Brady	5 Wren St	Edi	020	WC1H 9SE	3887834	Male	iPhone, 599 GBP	6pm 06/11/2009	UK
cf	(0.6)	(1.0)	(0.9)	(0.2)	(0.9)	(0.8)	(0.9)	(0.8)	(1.0)	(1.0)	(1.0)
t_4:	Robert	Brady	null	Ldn	020	WC1E 7HX	3887644	Male	necklace, 2,100 USD	1pm 06/11/2009	USA
cf	(0.7)	(1.0)	(0.0)	(0.5)	(0.7)	(0.3)	(0.7)	(0.8)	(1.0)	(1.0)	(1.0)

(b) Database D: An instance of schema tran

Figure 1: Example master data and database

and postal code; (3) the CFD φ_4 is a data standardization rule: if the first name is Bob, then it should be "normalized" as Robert; and (4) the MD ψ assures that for any tuple in D and any tuple in D_m, if they have the same last name and address, and moreover, if their first names are *similar*, then their phone and FN attributes can be identified.

Consider tuples t_3 and t_4 in D. The bank suspects that the two refer to the same person. If so, then these transaction records show that the same person made purchases in the UK and in the US at about the same time (taking into account the 5-hour time difference between the two countries). This indicates that a fraud has likely been committed.

Observe that t_3 and t_4 are quite different in their FN, city, St, post and Phn attributes. No rule allows us to identify the two directly. Nonetheless, they can indeed be matched by a sequence of *interleaved* matching and repairing operations:

(a) get a repair t'_3 of t_3 such that t'_3[city]=Ldn via the CFD φ_2, and t'_3[FN]=Robert by normalization with φ_4;
(b) match t'_3 with s_2 of D_m, to which ψ can be applied;
(c) as a result of the matching operation, get a repair t''_3 of t_3 by correcting t''_3[phn] with the master data s_2[tel];
(d) find a repair t'_4 of t_4 via the FD φ_3: since t''_3 and t_4 agree on their city and phn attributes, φ_3 can be applied. This allows us to enrich t_4[St] and fix t_4[post] by taking corresponding values from t''_3, which have been confirmed correct with the master data in step (c).

At this point t''_3 and t'_4 agree on every attribute in connection with personal information. It is now evident enough that they indeed refer to the same person; hence a fraud.

Observe that not only repairing helps matching (*e.g.,* from step (a) to (b)), but matching also helps us repair the data (*e.g.,* step (d) is doable only after the matching in (b)). □

This example tells us the following. (1) When taken together, record matching and data repairing perform much better than being treated as separate processes. (2) To make practical use of their interaction, matching and repairing operations should be *interleaved*. It does not help much to execute these processes consecutively one after another.

There has been a host of work on record matching (*e.g.,* [3, 6, 8, 19, 25, 37]; see [17, 26] for surveys) as well as on data repairing (*e.g.,* [4, 7, 10, 20, 21, 29, 39]). However, the problem of interleaving record matching and data repairing to improve the accuracy has not been well addressed.

Contributions. We investigate on cleaning data by *unifying* record matching and data repairing, and to provide a data cleaning solution that stresses accuracy.

(1) We investigate a new problem, stated as follows.

Given a database D, master data D_m, and data quality rules consisting of CFDs Σ and matching rules Γ, the *data cleaning problem* is to find a repair D_r of D such that (a) D_r is *consistent* (*i.e.,* satisfying the CFDs Σ), (b) no more tuples in D_r can be *matched* to master tuples in D_m by rules of Γ, and (c) D_r minimally differs from the original data D.

As opposed to record matching and data repairing, the data cleaning problem aims to fix errors in the data by unifying matching and repairing, and by leveraging master data. Here *master data* (*a.k.a.* reference data) is a single repository of high-quality data that provides various applications with a synchronized, consistent view of its core business entities [28]. It is being widely used in industry, supported by, *e.g.,* IBM, SAP, Microsoft and Oracle. To identify tuples from D and D_m, we use matching rules that are an extension of MDs [19] by supporting negative rules (*e.g.,* a male and female may not refer to the same person) [3, 37].

(2) We propose a uniform framework for data cleaning. We treat both CFDs and MDs as *cleaning rules*, which tell us how to fix errors. This yields a rule-based logical framework, which allows us to seamlessly interleave repairing and matching operations. To assure the accuracy of fixes, we make use of (a) the *confidence* placed by the user in the accuracy of the data, (b) *entropy* measuring the certainty of data, by the self-information of the data itself [12, 34], and (c) master data [28]. We distinguish three classes of fixes: (i) *deterministic* fixes for the unique solution to correct an error; (ii) *reliable* fixes for those derived using entropy; and (iii) *possible* fixes for those generated by heuristics. The former two are more accurate than possible fixes.

(3) We investigate fundamental problems associated with data cleaning via both matching and repairing. We show the following. (a) When CFDs and matching rules are taken together, the classical decision problems for dependencies, namely, the consistency and implication analyses, are NP-complete and coNP-complete, respectively. These problems have the same complexity as their counterparts for CFDs [18], *i.e.,* adding matching rules does not incur extra complexity. (b) The data cleaning problem is NP-complete. Worse still, it is approximation-hard, *i.e.,* it is beyond reach in practice to find a polynomial-time (PTIME) algorithm with a constant approximation ratio [35] unless P = NP. (c) It is more challenging to decide whether a data cleaning process terminates and whether it yields deterministic fixes: these problems are both PSPACE-complete.

(4) In light of the inherent complexity, we propose a three-phase solution consisting of three algorithms. (a) One algorithm identifies *deterministic fixes* that are accurate, based on confidence analysis and master data. (b) When confi-

dence is low or unavailable, we provide another algorithm to compute *reliable fixes* by employing information entropy, inferring evidence from data itself to improve accuracy. (c) To fix the remaining errors, we extend the heuristic based method [10] to find a consistent repair of the dirty data. These methods are complementary to each other, and can be used either alone or together.

(5) We experimentally evaluate the quality and scalability of our data cleaning methods with both matching and repairing, using real-life datasets (DBLP and hospital data from US Dept. of Health & Human Services). We find that our methods substantially outperform matching and repairing taken as separate processes in the accuracy of fixes, up to 15% and 30%, respectively. Moreover, deterministic fixes and reliable fixes are far more accurate than fixes generated by heuristic methods. Despite the high complexity of the cleaning problem, we also find that our algorithms scale reasonably well with the size of the data.

We contend that a unified process for repairing and matching is both important and feasible in practice, and that it should logically become part of data cleaning systems.

We remark that master data is desirable in the process. However, in its absence our approach can be adapted by interleaving (a) record matching in a single data table with MDs, as described in [17], and (b) data repairing with CFDs. While deterministic fixes would have lower accuracy, reliable fixes and heuristic fixes would not degrade substantially.

Organization. Section 2 reviews CFDs and extends MDs. Section 3 introduces the framework for data cleaning. Section 4 studies the fundamental problems for data cleaning. The two algorithms for data cleaning are provided in Sections 5 and 6 , respectively. Section 7 reports our experimental findings, followed by open issues in Section 8.

Related work. Record matching is also known as record linkage, entity resolution, merge-purge and duplicate detection (*e.g.,* [3,6,8,14,19,23,25,36,37]; see [17,26] for surveys). Matching rules are studied in [19, 25] (positive) and [3, 37] (negative). Data repairing was first studied in [4, 21]. A variety of constraints have been used to specify the consistency of data in data repairing, such as FDs [38], FDs and INDs [7], and CFDs [10, 18]. We employ CFDs, and extend MDs of [19] with negative rules. As remarked earlier, the problem of cleaning data by interleaving matching and repairing operations is not well addressed in previous work.

The consistency and implication problems have been studied for CFDs [18] and MDs [19]. We study these problems for MDs and CFDs put together. It is known that data repairing is NP-complete [7, 10]. We show that data cleaning via repairing and matching is NP-complete and approximation-hard. We also study the termination and determinism analyses of data cleaning, which are not considered in [7, 10].

Several repairing algorithms have been proposed [7, 10, 20, 21, 29, 39]. Heuristic methods are developed in [7, 10, 21], based on FDs and INDs [7], CFDs [18], and edit rules [21]. The methods of [7,10] employ confidence placed by users to guide a repairing process. Statistical inference is studied in [29] to derive missing values. To ensure the accuracy of repairs generated, [29,39] require to consult users. In contrast to the previous work, we (a) unify repairing and matching, (b) use confidence just to derive deterministic fixes, and (c) leverage master data and entropy to improve the accuracy. Closer to our work is [20], also based on master data. It differs from our work in the following. (i) While [20] aims to fix a *single* tuple via matching with editing rules (derived from MDs), we repair a *database* via *both* matching (MDs) and repairing (CFDs), a task far more challenging. (ii) While [20] only relies on confidence to warrant the accuracy, we use entropy analysis when the confidence is either low or unavailable.

There have also been efforts to interleave merging and matching operations [14, 23, 36, 37]. Among these, (1) [23] clusters data rather than repair data, and it does not update data to fix errors; and (2) [14, 36, 37] investigate record matching in the presence of error data, and advocate the need for data repairing to match records. The merge/fusion operations adopted there are more restrictive than updates (value modifications) considered in this work and data repairing in general. Further, when no matches are found, no merge or fusion is conducted, whereas this work may still repair data with CFDs.

There has also been a host of work on more general data cleaning: data transformation, which brings the data under a single common schema [30]. ETL tools (see [5] for a survey) provide sophisticated data transformation methods, which can be employed to merge data sets and repair data based on reference data. These are essentially orthogonal, but complementary, to data repairing and this work.

Information entropy measures the degree of uncertainty [12]: the less the entropy is, the more certain the data is. It has proved effective in, *e.g.,* database design, schema matching, data anonymization and data clustering [34]. We make a first effort to use it in data cleaning: we mark a fix reliable if its entropy is below a predefined threshold.

2. Data Quality Rules

Below we first review CFDs [18], which specify the consistency of data for data repairing. We then extend MDs [19] to match tuples across (a possibly dirty) database D and master data D_m. Both CFDs and MDs can be automatically discovered from data via profiling algorithms (*e.g.,* [9, 33]).

2.1 Conditional Functional Dependencies

Following [18], we define *conditional functional dependencies* CFDs on a relation schema R as follows.

A CFD φ defined on schema R is a pair $R(X \rightarrow Y,\ t_p)$, where (1) $X \rightarrow Y$ is a standard FD on R, referred to as *the FD embedded in* φ; and (2) t_p is a *pattern tuple* with attributes in X and Y, where for each A in $X \cup Y$, $t_p[A]$ is either a constant in the domain $\mathsf{dom}(A)$ of attribute A, or an unnamed variable '_' that draws values from $\mathsf{dom}(A)$.

We separate the X and Y attributes in t_p with '$\|$', and refer to X and Y as the LHS and RHS of φ, respectively.

Example 2.1: Recall the CFDs φ_1, φ_3 and φ_4 given in Example 1. These can be formally expressed as follows.

φ_1: $\mathsf{tran}([\mathsf{AC}] \rightarrow [\mathsf{city}],\ t_{p_1} = (131 \parallel \text{Edi}))$,
φ_3: $\mathsf{tran}([\mathsf{city}, \mathsf{phn}] \rightarrow [\mathsf{St}, \mathsf{AC}, \mathsf{post}],\ t_{p_3} = (\_, \_ \parallel \_, \_, \_))$
φ_4: $\mathsf{tran}([\mathsf{FN}] \rightarrow [\mathsf{FN}],\ t_{p_4} = (\text{Bob} \parallel \text{Robert}))$

Note that FDs are a special case of CFDs in which pattern tuples consist of only wildcards, *e.g.,* φ_3 given above. □

To give the formal semantics of CFDs, we use an operator $\asymp$ defined on constants and '_': $v_1 \asymp v_2$ if either $v_1 = v_2$, or one of v_1, v_2 is '_'. The operator $\asymp$ naturally extends to tuples, *e.g.,* (131, Edi) $\asymp$ (_, Edi) but (020, Ldn) $\not\asymp$ (_, Edi).

Consider an instance D of R. We say that D *satisfies* the

CFD φ, denoted by $D \models \varphi$, iff for *all* tuples t_1, t_2 in D, if $t_1[X] = t_2[X] \asymp t_p[X]$, then $t_1[Y] = t_2[Y] \asymp t_p[Y]$.

Example 2.2: Recall the tran instance D of Fig. 1(b) and the CFDs of Example 2.1. Observe that $D \not\models \varphi_1$ since tuple $t_1[\mathsf{AC}] = t_{p_1}[\mathsf{AC}]$, but $t_1[\mathsf{city}] \neq t_{p_1}[\mathsf{city}]$, *i.e.,* the single tuple t_1 violates φ_1. Similarly, $D \not\models \varphi_4$, as t_3 does not satisfy φ_4. Intuitively, φ_4 says that no tuple t can have $t[\mathsf{FN}]$ = Bob (it has to be changed to Robert). In contrast, $D \models \varphi_3$: there exist no distinct tuples in D that agree on city and phn. ☐

We say that an instance D of R *satisfies* a set Σ of CFDs, denoted by $D \models \Sigma$, if $D \models \varphi$ for each $\varphi \in \Sigma$.

2.2 Positive and Negative Matching Dependencies

Following [19,25], we define matching dependencies (MDs) in terms of a set Υ of similarity predicates, *e.g.,* q-grams, Jaro distance or edit distance (see *e.g.,* [17] for a survey).

We define positive MDs and negative MDs across a data relation schema R and a master relation schema R_m.

Positive MDs. A positive MD ψ on (R, R_m) is defined as:

$$\bigwedge_{j\in[1,k]}(R[A_j] \approx_j R_m[B_j]) \rightarrow \bigwedge_{i\in[1,h]}(R[E_i] \rightleftharpoons R_m[F_i]),$$

where (1) for each $j \in [1, k]$, A_j and B_j are attributes of R and R_m, respectively, with the same domain; similarly for E_i and F_i ($i \in [1, h]$); and (2) $\approx_j$ is a similarity predicate in Υ that is defined in the domain of $R[A_j]$ and $R_m[B_j]$. We refer to $\bigwedge_{j\in[1,k]}(R[A_j] \approx_j R_m[B_j])$ and $\bigwedge_{i\in[1,h]}(R[E_i] \rightleftharpoons R_m[F_i])$ as the LHS (premise) and RHS of ψ, respectively.

Note that MDs were originally defined on two unreliable data sources (see [19] for a detailed discussion of their dynamic semantics). In contrast, we focus on matching tuples across a dirty source D and a clean master relation D_m. To cope with this, we refine the semantics of MDs as follows.

For a tuple $t \in D$ and a tuple $s \in D_m$, if for each $j \in [1, k]$, $t[A_j]$ and $s[B_j]$ are similar, *i.e.,* $t[A_j] \approx_j s[B_j]$, then $t[E_i]$ is *changed* to $s[F_i]$, the clean master data, for each $i \in [1, h]$.

We say that an instance D of R *satisfies* the MD ψ *w.r.t. master data* D_m, denoted by $(D, D_m) \models \psi$, iff for *all* tuples t in D and *all* tuples s in D_m, if $t[A_j] \approx_j s[B_j]$ for $j \in [1, k]$, then $t[E_i] = s[F_i]$ for all $i \in [1, h]$.

Intuitively, $(D, D_m) \models \psi$ if no more tuples from D can be matched (and hence updated) with master tuples in D_m.

Example 2.3: Recall the MD ψ given in Example 1.1. Consider an instance D_1 of tran consisting of a single tuple t'_1, where $t'_1[\mathsf{city}]$ = Ldn and $t'_1[A] = t_1[A]$ for all the other attributes, for t_1 given in Fig. 1(b). Then $(D_1, D_m) \not\models \psi$, since $t'_1[\mathsf{FN}, \mathsf{phn}] \neq s_1[\mathsf{FN}, \mathsf{tel}]$ while (t'_1[LN, city, St, post] = s_1[LN, city, St, Zip] and $t'_1[\mathsf{FN}] \approx s_1[\mathsf{FN}]$. This suggests that we correct $t'_1[\mathsf{FN}, \mathsf{phn}]$ using the master data $s_1[\mathsf{FN}, \mathsf{tel}]$. ☐

Negative MDs. Along the same lines as [3, 37], we define a negative MD ψ^- as follows:

$$\bigwedge_{j\in[1,k]}(R[A_j] \neq R_m[B_j]) \rightarrow \bigvee_{i\in[1,h]}(R[E_i] \not\rightleftharpoons R_m[F_i]).$$

It states that for any tuple $t \in D$ and any tuple $s \in D_m$, if $t[A_j] \neq s[B_j]$ ($j \in [1, k]$), then t and s may not be identified.

Example 2.4: A negative MD defined on (tran, card) is:

$$\psi_1^-: \mathsf{tran}[\mathsf{gd}] \neq \mathsf{card}[\mathsf{gd}] \rightarrow \bigvee_{i\in[1,7]}(\mathsf{tran}[A_i] \not\rightleftharpoons \mathsf{card}[B_i]),$$

where (A_i, B_i) ranges over (FN, FN), (LN, LN), (St, St), (AC, AC), (city, city), (post, zip) and (phn, tel). It says that a male and a female may not refer to the same person. ☐

We say that an instance D of R *satisfies* the negative MD ψ^- *w.r.t. master data* D_m, denoted by $(D, D_m) \models \psi^-$, if for *all* tuples t in D and *all* tuples s in D_m, if $t[A_j] \neq s[B_j]$ for all $j \in [1, k]$, then there exists $i \in [1, h]$ such that $t[E_i] \neq s[F_i]$.

An instance D of R *satisfies* a set Γ of (positive, negative) MDs *w.r.t. master data* D_m, denoted by $(D, D_m) \models \Gamma$, if $(D, D_m) \models \psi$ for all $\psi \in \Gamma$.

Normalized CFDs and MDs. Given a CFD (resp. MD) ξ, we use LHS(ξ) and RHS(ξ) to denote the LHS and RHS of ξ, respectively. It is called *normalized* if $|\text{RHS}(\xi)| = 1$, *i.e.,* its right-hand side consists of a single attribute (resp. attribute pair). As shown by [18,19], every CFD ξ (resp. MD) can be expressed as an equivalent set S_ξ of CFDs (resp. MDs), such that the cardinality of S_ξ is bounded by the size of RHS(ξ).

For instance, the CFDs φ_1, φ_2 and φ_4 of Example 1.1 are normalized. While the CFD φ_3 is not normalized, it can be converted to an equivalent set of CFDs of the form ([city, phn] $\rightarrow A_i, t_{p_i}$), where A_i ranges over St, AC and post, and t_{p_i} consists of wildcards only; similarly for the MD ψ.

We consider normalized CFDs (MDs) only in the sequel.

3. A Uniform Framework for Data Cleaning

We propose a rule-based framework for data cleaning. It treats CFDs and MDs uniformly as *cleaning rules*, which tell us how to fix errors, and seamlessly interleaves matching and repairing operations (Section 3.1). Using cleaning rules we introduce a tri-level data cleaning solution, which generates fixes with various levels of accuracy, depending on the information available about the data (Section 3.2).

Consider a (possibly dirty) relation D of schema R, a master relation D_m of schema R_m, and a set $\Theta = \Sigma \cup \Gamma$, where Σ is a set of CFDs on R, and Γ is a set of MDs on (R, R_m).

3.1 A Rule-based Logical Framework

We first state the data cleaning problem, and then define cleaning rules derived from CFDs and MDs.

Data cleaning. The *data cleaning problem*, referred to as DCP, takes as input D, D_m and Θ; it is to compute a *repair* D_r of D, *i.e.,* another database such that (a) $D_r \models \Sigma$, (b) $(D_r, D_m) \models \Gamma$, and (c) $\mathsf{cost}(D_r, D)$ is minimum.

Intuitively, (a) the repair D_r of D should be *consistent*, (b) no more tuples in D_r can be *matched* to master data, and (c) D_r is accurate and is close to the original data D.

Using the quality model of [10], we define $\mathsf{cost}(D_r, D)$ as:

$$\sum_{t\in D}\sum_{A\in\mathsf{attr}(R)} t(A).\mathsf{cf} * \frac{\mathsf{dis}_A(t[A], t'[A])}{\mathsf{max}(|t[A]|, |t'[A]|)}$$

where (a) tuple $t' \in D_r$ is the repair of tuple $t \in D$, (b) $\mathsf{dis}_A(v, v')$ is the distance between values $v, v' \in \mathsf{dom}(A)$; the smaller the distance is, the closer the two values are to each other; (c) $|t[A]|$ denotes the size of $t[A]$; and (d) $t[A]$.cf is the *confidence* placed by the user in the accuracy of the attribute $t[A]$ (see the cf rows in Fig. 1(b)).

This quality metric says that the higher the confidence of the attribute $t[A]$ is and the more distant v' is from v, the more costly the change is. Thus, the smaller $\mathsf{cost}(D_r, D)$ is, the more accurate and closer to the original data D_r is. We use $\mathsf{dis}(v, v')/\mathsf{max}(|v|, |v'|)$ to measure the similarity of v and v' to ensure that longer strings with 1-character difference are closer than shorter strings with 1-character difference.

As remarked in [10], confidence can be derived via provenance analysis, which can be reinforced by recent work on determining the reliability of data sources (*e.g.,* [15]).

Cleaning rules. A variety of integrity constraints have been studied for data repairing (*e.g.,* [7,10,18,38]). As observed by [20], while there constraints help us determine whether data is dirty or not, *i.e.,* whether errors are present in the data, they do not tell us how to correct the errors.

To make better practical use of constraints in data cleaning, we define *cleaning rules*, which tell us what attributes should be updated and to what value they should be changed. From each MD in Γ and each CFD in Σ, we derive a cleaning rule as follows, based on fuzzy logic [27].

(1) MDs. Consider an MD $\psi = \bigwedge_{j\in[1,k]}(R[A_j] \approx_j R_m[B_j]) \rightarrow (R[E] \rightleftharpoons R_m[F])$. The *cleaning rule* derived from ψ, denoted by γ_ψ, *applies* a master tuple $s \in D_m$ to a tuple $t \in D$ if $t[A_j] \approx_j s[B_j]$ for each $j \in [1,k]$. It *updates* t by letting (a) $t[E] := s[F]$ and (b) $t[A].\mathsf{cf} := d$, where d is the minimum $t[A_j].\mathsf{cf}$ for all $j \in [1,k]$ if $\approx_j$ is '='.

That is, γ_ψ corrects $t[E]$ with clean master value $s[F]$, and infers the new confidence of $t[E]$ following fuzzy logic [27].

(2) Constant CFDs. Consider a CFD $\varphi_c = R(X \rightarrow A,\ t_{p_1})$, where $t_{p_1}[A]$ is a *constant*. The *cleaning rule* derived from φ_c *applies to* a tuple $t \in D$ if $t[X] \asymp t_{p_1}[X]$ but $t[A] \neq t_{p_1}[A]$. It *updates* t by letting (a) $t[A] := t_{p_1}[A]$, and (b) $t[A].\mathsf{cf} = d$, where d is the minimum $t[A'].\mathsf{cf}$ for all $A' \in X$. That is, the rule corrects $t[A]$ with the constant in the CFD.

(3) Variable CFDs. Consider a CFD $\varphi_v = (Y \rightarrow B,\ t_{p_2})$, where $t_{p_2}[B]$ is a wildcard '_'. The *cleaning rule* derived from φ_v is used to *apply* a tuple $t_2 \in D$ to another tuple $t_1 \in D$, where $t_1[Y] = t_2[Y] \asymp t_{p_2}[Y]$ but $t_1[B] \neq t_2[B]$. It *updates* t_1 by letting (a) $t_1[B] := t_2[B]$, and (b) $t_1[B].\mathsf{cf}$ be the minimum $t_1[B'].\mathsf{cf}$ and $t_2[B'].\mathsf{cf}$ for all $B' \in Y$.

While cleaning rules derived from MDs are similar to editing rules of [20], rules derived from (constant or variables) CFDs are not studied in [20]. We use confidence information and infer new confidences based on fuzzy logic [27].

Embedding negative MDs. Recall negative MDs from Section 2.2. The example below tells us that negative MDs can be converted to equivalent positive MDs. As a result, there is no need to treat them separately.

Example 3.1: Consider the MD ψ in Example 1.1 and the negative MD ψ^- in Example 2.4. We define ψ' by incorporating the premise (gd) of ψ^- into the premise of ψ:

$$\psi'\colon\ \mathsf{tran}[\mathsf{LN},\mathsf{city},\mathsf{St},\mathsf{post},\mathsf{gd}] = \mathsf{card}[\mathsf{LN},\mathsf{city},\mathsf{St},\mathsf{zip},\mathsf{gd}] \wedge \mathsf{tran}[\mathsf{FN}] \approx \mathsf{card}[\mathsf{FN}] \rightarrow \mathsf{tran}[\mathsf{FN},\mathsf{phn}] \rightleftharpoons \mathsf{card}[\mathsf{FN},\mathsf{tel}].$$

Then no tuples with different genders can be identified as the same person, which is precisely what ψ^- is to enforce. In other words, the positive MD ψ' is equivalent to the positive MD ψ and the negative MD ψ^-. □

Indeed, it suffices to consider only positive MDs.

Proposition 3.1: *Given a set Γ_m^+ of positive MDs and a set Γ_m^- of negative MDs, there exists an algorithm that computes a set Γ_m of positive MDs in $O(|\Gamma_m^+||\Gamma_m^-|)$ time such that Γ_m is equivalent to $\Gamma_m^+ \cup \Gamma_m^-$.* □

A uniform framework. By treating both CFDs and MDs as cleaning rules, one can uniformly interleave matching and repairing operations, to facilitate their interactions.

Example 3.2: As shown in Example 1.1, to clean tuples t_3 and t_4 of Fig. 1(b), one needs to interleave matching and repairing operations. These can be readily done by using cleaning rules derived from φ_2, φ_4, ψ and φ_3. Indeed, the cleaning process described in Example 1.1 is actually carried out by applying these rules. There is no need to distinguish between matching and repairing in the cleaning process. □

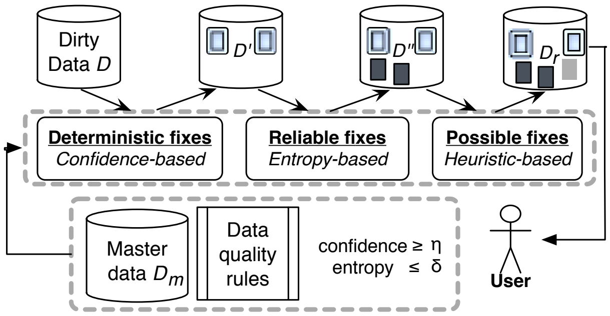


Figure 2: Framework Overview

3.2 A Tri-level Data Cleaning Solution

Based on cleaning rules, we develop a data cleaning system UniClean. It takes as input a dirty relation D, a master relation D_m, a set of cleaning rules derived from Θ, as well as thresholds $\eta, \delta \in [0,1]$ set by the users for confidence and entropy, respectively. It generates a repair D_r of D with a small $\mathsf{cost}(D_r, D)$, such that $D_r \models \Sigma$ and $(D_r, D_m) \models \Gamma$.

As opposed to previous repairing systems [7,10,20,21,29,39], UniClean generates fixes by unifying matching and repairing, via cleaning rules. Further, it stresses the accuracy by distinguishing these fixes with three levels of accuracy. Indeed, various fixes are found by three algorithms executed one after another, as shown in Fig. 2 and illustrated below.

(1) Deterministic fixes based on confidences. The first algorithm identifies erroneous attributes $t[A]$ to which there exists a unique fix, referred to as a *deterministic fix*, when some attributes of t are accurate. It fixes those errors based on confidence: it uses a cleaning rule to update $t[A]$ *only if* certain attributes of t have confidence above the threshold η. It is evident that such fixes are accurate up to η.

(2) Reliable fixes based on entropy. For attributes with low or unavailable confidence, we correct them based on the relative certainty of the data, measured by entropy. Entropy has proved effective in data transmission [24] and compression [40], among other things. We use entropy to clean data: we apply a cleaning rule γ to update an erroneous attribute $t[A]$ only if the entropy of γ for certain attributes of t is below the threshold δ. Fixes generated via entropy are accurate to a certain degree, and are marked as *reliable fixes*.

(3) Possible fixes. Not all errors can be fixed in the first two phases. For the remaining errors, we adopt heuristic methods to generate fixes, referred to as *possible fixes*. To this end we extend the method of [10], by supporting cleaning rules derived from both CFDs and MDs. Fixes produced in the first two phases remain unchanged at this step. Notably, the other heuristic methods can also be (possibly) adapted.

At the end of the process, fixes are marked with three distinct signs, indicating deterministic, reliable and possible, respectively. We shall present methods based on confidence and entropy in Sections 5 and 6, respectively. Due to space limitations, we opt to omit the algorithm of possible fixes. For interesting readers, please refer to [10] for more details.

4. Fundamental Problems for Data Cleaning

We now investigate fundamental problems associated with data cleaning. We first study the consistency and implication problems for CFDs and MDs taken together, from which cleaning rules are derived. We then establish the complexity bounds of the data cleaning problem as well as its termination and determinism analyses. These problems are not only of theoretical interest, but are also important to the development of data cleaning algorithms. The main conclusion of this section is that data cleaning via matching and repairing is inherently difficult: all these problems are intractable.

Consider a relation D of schema R, a master data D_m of schema R_m, and a set $\Theta = \Sigma \cup \Gamma$, where Σ is a set of CFDs defined on R, and Γ is a set of MDs defined on (R, R_m).

4.1 Reasoning about Data Quality Rules

There are two classical problems for data quality rules.

The *consistency problem* is to determine, given D_m and $\Theta = \Sigma \cup \Gamma$, whether there exists a nonempty instance D of R such that $D \models \Sigma$ and $(D, D_m) \models \Gamma$.

Intuitively, this is to determine whether the rules in Θ are dirty themselves. The practical need for the consistency analysis is evident: it does not make sense to derive cleaning rules from Θ before Θ is assured consistent itself.

We say that Θ *implies* another CFD (resp. MD) ξ, denoted by $\Sigma \models \xi$, if for any instance D of R, whenever $D \models \Sigma$ and $(D, D_m) \models \Gamma$, then $D \models \xi$ (resp. $(D, D_m) \models \xi$).

The *implication problem* is to determine, given D_m, Σ and another CFD (or MD) ξ, whether $\Sigma \models \xi$.

Intuitively, the implication analysis helps us find and remove redundant rules from Σ, *i.e.,* those that are a logical consequence of other rules in Σ, to improve performance.

These problems have been studied for CFDs and MDs separately. It is known that the consistency problem for MDs is trivial: any set of MDs is consistent [19]. In contrast, there exist CFDs that are inconsistent, and the consistency analysis of CFDs is NP-complete [18]. It is also known that the implication problem for MDs and CFDs is in quadratic time [19] and coNP-complete [18], respectively.

We show that these problems for CFDs and MDs put together have the same complexity as their CFDs counterparts. That is, adding MDs to CFDs does not make our lives harder.

Theorem 4.1: *For CFDs and MDs put together, the consistency problem is NP-complete, and the implication problem is coNP-complete (when ξ is either a CFD or an MD).* □

Proof: The upper bounds are verified by establishing a small model property. The lower bounds follow from the intractability for their CFD counterparts, a special case. □

In the rest of the paper we consider only collections Σ of CFDs and MDs that are consistent.

4.2 Analyzing the Data Cleaning Problem

Recall the data cleaning problem (DCP) from Section 3.

Complexity bounds. One wants to know how costly it is to compute a repair D_r. Below we show that it is intractable to decide whether there exists D_r with $\mathsf{cost}(D_r, D)$ below a predefine bound. Worse still, it is infeasible in practice to find PTIME approximation algorithm with performance guarantee. Indeed, the problem is not even in APX, the class of problems that allow PTIME approximation algorithms with approximation ratio bounded by a constant.

Theorem 4.2: *(a) The data cleaning problem (DCP) is NP-complete. (b) Unless P = NP, for any constant ϵ, there exists no* PTIME *ϵ-approximation algorithm for DCP.* □

Symbols	Semantics
$\Theta = \Sigma \cup \Gamma$	A set Σ of CFDs and a set Γ of MDs
η, δ_1, δ_2	Confidence threshold, update threshold, and entropy threshold, respectively
ρ	Selection operator in relational algebra
π	Projection operator in relational algebra
$\Delta(\bar{y})$	The set $\{t \mid t \in D, t[Y] = \bar{y}\}$ for each $\bar{y}$ in $\pi_Y(\rho_{Y \asymp t_p[Y]} D)$ *w.r.t.* CFD $(Y \rightarrow B,\ t_p)$

Table 1: Summary of notations

Proof: (a) The upper bound is verified by giving an NP algorithm. The lower bound is by reduction from 3SAT [35]. (b) This is verified by reduction from 3SAT, using gap techniques [35]. Given any constant ϵ, we show that there exists an algorithm with approximation ratio ϵ for DCP iff there is a PTIME algorithm for deciding 3SAT. □

It is known that data repairing alone is NP-complete [10]. Theorem 4.2 tells us that when matching with MDs is incorporated, the problem is intractable and approximation-hard.

Termination and determinism analyses. There are two natural questions about rule-based data cleaning methods such as the one proposed in Section 3. (a) The *termination problem* is to determine whether a rule-based process stops. That is, it reaches a *fixpoint*, such that no cleaning rules can be further applied. (b) The *determinism problem* asks whether all terminating cleaning processes end up with the same repair, *i.e.,* all of them reach a *unique* fixpoint.

The need for studying these problems is evident. A rule-based process is often *non-deterministic*: multiple rules can be applied at the same time. We want to know whether the output of the process is independent of the order of the rules applied. Worse, it is known that even for repairing only, a rule-based method may lead to an *infinite* process [10].

Example 4.1: Consider the CFD $\varphi_1 = \mathsf{tran}([\mathsf{AC}] \rightarrow [\mathsf{city}], t_{p_1} = (131 \parallel \text{Edi}))$ given in Example 2.1, and another CFD $\varphi_5 = \mathsf{tran}([\mathsf{post}] \rightarrow [\mathsf{city}], t_{p_5} = (\text{EH8 9AB} \parallel \text{Ldn}))$. Consider D_1 consisting of a single tuple t_2 given in Fig. 1. Then a repairing process for D_1 with φ_1 and φ_5 may fail to terminate: it changes $t_2[\mathsf{city}]$ to Edi and Ldn back and forth. □

No matter how important, it is beyond reach in practice to find efficient solutions to these two problems.

Theorem 4.3: *The termination and determinism problems are both PSPACE-complete for rule-based data cleaning.* □

Proof: We verify the lower bound of these problems by reduction from the halting problem for linear bound automata, which is PSPACE-complete [2]. We show the upper bound by providing an algorithm for each of the two problems, which uses polynomial space in the size of input. □

5. Deterministic Fixes with Data Confidence

As shown in Fig. 2, system UniClean first identifies deterministic fixes based on confidence analysis and master data. In this section we define deterministic fixes (Section 5.1), and present an efficient algorithm to find them (Section 5.2).

In Table 1 we summarize some notations to be used in this Section and Section 6.

5.1 Deterministic Fixes

We define deterministic fixes *w.r.t.* a *confidence threshold* η determined by domain experts. When η is high enough, *e.g.,* if it is close to 1, an attribute $t[A]$ is assured correct if $t[A].\mathsf{cf} \geq \eta$. We refer to such attributes as *asserted* attributes. Recall from Section 3 the definition of cleaning rules derived from MDs and CFDs. In the first phase of UniClean, we apply a cleaning rule γ to tuples in a database D only when the attributes in the premise (*i.e.,* LHS) of γ are all asserted. We say that a fix is *deterministic w.r.t.* γ and η if it is generated as follows, based on how γ is derived.

(1) From an MD $\psi = \bigwedge_{j\in[1,k]}(R[A_j] \approx_j R_m[B_j]) \rightarrow (R[E] \rightleftharpoons R_m[F])$. Suppose that γ applies a tuple $s \in D_m$ to a tuple $t \in D$, and generates a fix $t[E] := s[F]$ (see Section 3.1). Then the fix is *deterministic* if $t[A_j].\mathsf{cf} \geq \eta$ for all $j \in [1,k]$ and moreover, $t[E].\mathsf{cf} < \eta$. That is, $t[E]$ is changed to the master value $s[F]$ only if (a) all the premise attributes $t[A_j]$'s are asserted, and (b) $t[E]$ is not yet asserted.

(2) From a constant CFD $\varphi_c = R(X \rightarrow A,\ t_{p_1})$. Suppose that γ applies to a tuple $t \in D$ and changes $t[A]$ to the constant $t_{p_1}[A]$ in φ_c. Then the fix is *deterministic* if $t[A_i].\mathsf{cf} \geq \eta$ for all $A_i \in X$ and $t[A].\mathsf{cf} < \eta$.

(3) From a variable CFD $\varphi_v = (Y \rightarrow B,\ t_p)$. For each $\bar{y}$ in $\pi_Y(\rho_{Y\asymp t_p[Y]}D)$, we define $\Delta(\bar{y})$ to be the set $\{t \mid t \in D, t[Y] = \bar{y}\}$, where π and ρ are the projection and selection operators, respectively, in relational algebra [1]. That is, for all t_1, t_2 in $\Delta(\bar{y})$, $t_1[Y] = t_2[Y] = \bar{y} \asymp t_p[Y]$.

Suppose that γ applies a tuple t_2 in $\Delta(\bar{y})$ to another t_1 in $\Delta(\bar{y})$ for some $\bar{y}$, and changes $t_1[B]$ to $t_2[B]$. Then the fix is *deterministic* if (a) for all $B_i \in Y$, $t_1[B_i].\mathsf{cf} \geq \eta$ and $t_2[B_i].\mathsf{cf} \geq \eta$, (b) $t_2[B].\mathsf{cf} \geq \eta$, and moreover, (c) t_2 is the only tuple in $\Delta(\bar{y})$ with $t_2[B].\mathsf{cf} \geq \eta$ (hence $t_1[B].\mathsf{cf} < \eta$). That is, all the premise attributes of γ are asserted, and $t_2[B]$ is the only value of B-attribute in $\Delta(\bar{y})$ that is assumed correct, while $t_1[B]$ is suspected erroneous.

As observed by [20], when data quality rules and asserted attributes are assured correct, the fixes generated are unique (called "certain" in [20]). While [20] only considers MDs, the observation remains intact for CFDs and MDs.

Note that when an attribute $t[A]$ is updated by a deterministic fix, its confidence $t[A].\mathsf{cf}$ is upgraded to be the minimum of the confidences of the premise attributes (see Section 3.1). As a result, $t[A]$ also becomes asserted, since all premise attributes have confidence values above η. In turn $t[A]$ can be used to generate deterministic fixes for other attributes in the cleaning process. In other words, the process for finding deterministic fixes in a database D is *recursive.*

Nevertheless, in the rest of the section we show that deterministic fixes can be found in PTIME, stated as follows.

Theorem 5.1: *Given master data D_m and a set Θ of CFDs and MDs, all deterministic fixes in a relation D can be found in $O(|D||D_m|\mathsf{size}(\Theta))$ time, where $\mathsf{size}(\Theta)$ is Θ's length.* □

5.2 Confidence-based Data Cleaning

We next introduce the algorithm, followed by the indexing structures and procedures that it employs.

Algorithm. The algorithm, denoted by cRepair, is shown in Fig. 3. It takes as input CFDs Σ, MDs Γ, master data D_m, dirty data D, and a confidence threshold η. It returns a partially cleaned repair D' with deterministic fixes marked.

The algorithm first initializes variables and indexing structures (lines 1–6). It then recursively computes deterministic fixes (lines 7–15), by invoking procedures vCFDInfer (line 12), cCFDInfer (line 13), or MDInfer (line 14), for the rules derived from variable CFDs, constant CFDs, or MDs, respectively. These indexing structures and procedures will be discussed immediately. It terminates when no more deterministic fixes can be found (line 15). Finally, a partially cleaned database D' is returned in which all deterministic fixes are identified (line 16).

Algorithm cRepair

Input: CFDs Σ, MDs Γ, master data D_m, dirty data D, and confidence threshold η.
Output: A partial repair D' of D with deterministic fixes.

1. $D' := D$; $\mathsf{H}_\xi := \emptyset$ for each variable CFD $\xi \in \Sigma$;
2. **for** each $t \in D'$ **do**
3. $\mathsf{Q}[t] := \emptyset$; $\mathsf{P}[t] := \emptyset$;
4. $\mathsf{count}[t, \xi] := 0$ for each $\xi \in \Sigma \cup \Gamma$;
5.   **for** each attribute $A \in \mathsf{attr}(\Sigma \cup \Gamma)$ **do**
6.   **if** $t[A].\mathsf{cf} \geq \eta$ **then** $\mathsf{update}(t, A)$;
7. **repeat**
8.   **for** each tuple $t \in D'$ **do**
9.   **while** $\mathsf{Q}[t]$ is not empty **do**
10. $\xi := \mathsf{Q}[t].\mathsf{pop}()$;
11.   **case** ξ of
12.   (1) variable CFD: $D' := \mathsf{vCFDInfer}(t,\ \xi,\ \eta)$;
13.   (2) constant CFD: $D' := \mathsf{cCFDInfer}(t,\ \xi,\ \eta)$;
14.   (3) MD: $D' := \mathsf{MDInfer}(t,\ \eta,\ D_m,\ \xi)$;
15. **until** $\mathsf{Q}[t']$ is empty for any $t' \in D'$;
16. **return** D'.

Figure 3: Algorithm cRepair

Indexing structures. The algorithm uses the following indexing structures, to improve performance.

Hash tables. We maintain a hash table for each variable CFD $\varphi = R(Y \rightarrow B,\ t_p)$, denoted as H_φ. Given a $\bar{y} \in \rho_{Y\asymp t_p[Y]}(D)$ as the key, it returns a pair $(\mathsf{list}, \mathsf{val})$ as the value, *i.e.,* $\mathsf{H}(\bar{y}) = (\mathsf{list}, \mathsf{val})$, where (a) list consists of all the tuples t in $\Delta(\bar{y})$ such that $t[B_i].\mathsf{cf} \geq \eta$ for each attribute $B_i \in Y$, and (b) val is $t[B]$ if it is the only item in $\Delta(\bar{y})$ with $t[B].\mathsf{cf} \geq \eta$; otherwise, val is nil. Notably, there exist no two t_1, t_2 in $\Delta(\bar{y})$ such that $t_1[B] \neq t_2[B]$, $t_1[B].\mathsf{cf} \geq \eta$ and $t_2[B].\mathsf{cf} \geq \eta$, if the confidence placed by the users is correct.

Queues. We maintain for each tuple t a queue of rules that can be applied to t, denoted as $\mathsf{Q}[t]$. More specifically, $\mathsf{Q}[t]$ contains all rules $\xi \in \Theta$, where $t[C].\mathsf{cf} \geq \eta$ for all attributes C in LHS(ξ). That is, the premise of ξ is asserted in t.

Hash sets. For each tuple $t \in D$, $\mathsf{P}[t]$ stores the set of variable CFDs $\varphi \in \mathsf{Q}[t]$ such that $\mathsf{H}_\varphi(t[\text{LHS}(\varphi)]).\mathsf{val} = \mathsf{nil}$, *i.e.,* no B attribute in $\Delta(t[\text{LHS}(\varphi)])$ has a high confidence.

Counters. For each tuple $t \in D$ and each rule $\xi \in \Theta$, $\mathsf{count}[t, \xi]$ is a counter that maintains the number of current values of the attributes $C \in \text{LHS}(\xi)$ such that $t[C].\mathsf{cf} \geq \eta$.

Procedures. We present the procedures used in cRepair.

update. Given a new deterministic fix for $t[A]$, it propagates the change and finds other deterministic fixes with $t[A]$. (a) For each rule ξ, if $A \in \text{LHS}(\xi)$, $\mathsf{count}[t, \xi]$ is increased by 1 since one more attribute becomes asserted. (b) If all attributes in LHS(ξ) are asserted, ξ is inserted into the queue $\mathsf{Q}[t]$. (c) For a variable CFD $\xi' \in \mathsf{P}[t]$, if RHS(ξ') is A and $\mathsf{H}_{\xi'}(t[\text{LHS}(\xi')]).\mathsf{val} = \mathsf{nil}$, then the newly asserted $t[A]$ makes it possible for those tuples in $\mathsf{H}_{\xi'}(t[\text{LHS}(\xi')]).\mathsf{list}$ to have a deterministic fix. Thus ξ' is removed from $\mathsf{P}[t]$ and added to $\mathsf{Q}[t]$, to be examined.

vCFDInfer. Given a tuple t, a variable CFD ξ and the confidence threshold η, it finds a deterministic fix for t by applying ξ if it exists. If the tuple t and the pattern tuple $t_{(p,\xi)}$ match on their LHS(ξ) attributes, we have the follows.

(a) If $t[\text{RHS}(\xi)].\mathsf{cf} \geq \eta$ and no B-attribute value in

$\mathsf{H}_\xi(t[\text{LHS}(\xi)]).\mathsf{list}$ is asserted, it takes $t[\text{RHS}(\xi)]$ as the value of the B attributes in the set. The change is propagated by procedure update.

(b) If $t[\text{RHS}(\xi)] < \eta$ and there is a B-attribute value val in $\mathsf{H}_\xi(t[\text{LHS}(\xi)]).\mathsf{list}$ with a high confidence, a deterministic fix is made by changing $t[\text{RHS}(\xi)]$ to val. The change is propagated via update.

(c) If $t[\text{RHS}(\xi)] < \eta$ but no asserted B-attributes in $\mathsf{H}_\xi(t[\text{LHS}(\xi)]).\mathsf{list}$, we cannot make a deterministic fix at the moment. Tuple t is added to $\mathsf{H}_\xi(t[\text{LHS}(\xi)]).\mathsf{list}$ and $\mathsf{P}[t]$, for later checking.

cCFDInfer *and* MDInfer. The first one takes as input a tuple t, a constant CFD ξ and the threshold η. The second one takes as input t, η, master data D_m and an MD ξ. They find deterministic fixes by applying the rules derived from ξ, as described earlier. The changes made are propagated by invoking $\mathsf{update}(t, \text{RHS}(\xi))$.

We next show by example how algorithm cRepair works.

Example 5.1: Consider the master data D_m and the relation D in Fig. 1. Assume Θ consists of rules ξ_1, ξ_2 and ξ_3, derived from the CFDs φ_1, φ_3 and the MD ψ of Example 1.1, respectively. Let the threshold η be 0.8. Using Θ and D_m, it finds deterministic fixes for $t_1, t_2 \in D$ *w.r.t.* η as follows.

(1) After initialization (lines 1–6), we have: (a) $\mathsf{H}_{\xi_2} = \emptyset$; (b) $\mathsf{Q}[t_1] = \{\xi_1\}$, $\mathsf{Q}[t_2] = \{\xi_2\}$; (c) $\mathsf{P}[t_1] = \mathsf{P}[t_2] = \emptyset$; and (d) $\mathsf{count}[t_1, \xi_1] = 1$, $\mathsf{count}[t_1, \xi_2] = 0$, $\mathsf{count}[t_1, \xi_3] = 3$, $\mathsf{count}[t_2, \xi_1] = 0$, $\mathsf{count}[t_2, \xi_2] = 2$, and $\mathsf{count}[t_2, \xi_3] = 2$.

(2) After $\xi_2 \in \mathsf{Q}[t_2]$ is checked (line 12), we have $\mathsf{Q}[t_2] = \emptyset$, $\mathsf{P}[t_2] = \{\xi_2\}$, and $\mathsf{H}_{\xi_2}(t_2[\mathsf{city}, \mathsf{phn}]) = (\{t_2\}, \mathsf{nil})$.

(3) After $\xi_1 \in \mathsf{Q}[t_1]$ is applied (line 13), $\mathsf{Q}[t_1] = \{\xi_3\}$, $\mathsf{count}[t_1, \xi_2] = 1$ and $\mathsf{count}[t_1, \xi_3] = 4$. This step finds a deterministic fix $t_1[\mathsf{city}]$:= Edi. It upgrades $t_1[\mathsf{city}].\mathsf{cf}$:=0.8.

(4) When $\xi_3 \in \mathsf{Q}[t_1]$ is used (line 14), it makes a fix $t_1[\mathsf{phn}]$:= $s_1[\mathsf{tel}]$, and lets $t_1[\mathsf{phn}].\mathsf{cf}$ = 0.8. Now we have $\mathsf{Q}[t_1] = \{\xi_2\}$ and $\mathsf{count}[t_1, \xi_2] = 2$.

(5) When $\xi_2 \in \mathsf{Q}[t_1]$ is used (line 14), it finds a deterministic fix by letting $t_2[\mathsf{St}] = t_1[\mathsf{St}]$:= 10 Oak St, and $t_2[\mathsf{St}].\mathsf{cf}$:= 0.8. Now we obtain $\mathsf{Q}[t_1] = \emptyset$ and $\mathsf{P}[t_2] = \emptyset$.

(6) Finally, the process terminates since $\mathsf{Q}[t_1] = \mathsf{Q}[t_2] = \emptyset$.

Similarly, for tuples $t_3, t_4 \in D$, cRepair finds a deterministic fix by letting $t_3[\mathsf{city}]$:= Ldn and $t_3[\mathsf{city}].\mathsf{cf}$:= 0.8. □

Suffix trees for similarity checking of MDs. For cleaning rules derived from MDs, we need to conduct *similarity checking*, to which traditional indexing techniques are not directly applicable. To cope with this, we develop a technique upon suffix trees [13]. The measure of similarity adopted is the length of the *longest common substring* of two strings. Generalized suffix trees are built for the blocking process with all the strings in the active domain. When querying the k-most similar strings of v of length $|v|$, we can extract the subtree T of suffix tree that only contains branches related to v that contains at most $|v|^2$ nodes; and by traversing T to find the k-most similar strings. In this way, we can identify k similar values from D_m in $O(k|v|^2)$ time, which reduces the search space from $|D_m|$ to a constant number k of tuples. Our experimental study verifies that the technique significantly improves the performance.

Complexity. Note that for each CFD in Σ, each tuple t in D is examined at most twice. For each MD, each tuple $t \in D$ is checked at most $|D_m|$ times. From these it follows that algorithm cRepair is in $O(|D||D_m|\mathsf{size}(\Sigma \cup \Gamma))$ time. With the optimization techniques above, the time complexity of cRepair is reduced to $O(|D|\mathsf{size}(\Sigma \cup \Gamma))$.

6. Reliable Fixes with Information Entropy

Deterministic fixes may not exist for some attributes, *e.g.,* when their confidences are low or unreliable. To find accurate fixes for these attributes, UniClean looks for evidence from data itself, using entropy to measure the degree of certainty. Below we first define entropy for data cleaning (Section 6.1), and then present an algorithm to find reliable fixes based on entropy (Section 6.2). Finally we present an indexing structure that underlines the algorithm (Section 6.3).

6.1 Measuring Certainty with Entropy

We start with an overview of the standard information entropy, and then define entropy for resolving conflicts.

Entropy. The entropy of a discrete random variable $\mathcal{X}$ with possible values $\{x_1, \ldots, x_n\}$ is defined as [12, 34]:

$$\mathcal{H}(\mathcal{X}) = \Sigma_{i=1}^{n}(p_i * \log 1/p_i),$$

where p_i is the probability of x_i for $i \in [1, n]$. The entropy measures the degree of the certainty of the value of $\mathcal{X}$: when $\mathcal{H}(\mathcal{X})$ is sufficiently small, it is highly accurate that the value of $\mathcal{X}$ is the x_j having the largest probability p_j. The less $\mathcal{H}(\mathcal{X})$ is, the more accurate the prediction is.

Entropy for variable CFDs. We use entropy to resolve data conflicts. Consider a CFD $\varphi = R(Y \to B, t_p)$ defined on a relation D, where $t_p[B]$ is a wildcard. Note that a deterministic fix may not exist when, *e.g.,* there are t_1, t_2 in $\Delta(\bar{y})$ (see Table 1) such that $t_1[B] \neq t_2[B]$ but both have high confidence. Indeed, using the cleaning rule derived from φ, one may either let $t_1[B] := t_2[B]$ by applying t_2 to t_1, or let $t_2[B] := t_1[B]$ by applying t_1 to t_2.

To find an accurate fix, we define the entropy of φ for $Y = \bar{y}$, denoted by $\mathcal{H}(\varphi|Y = \bar{y})$, as

$$\mathcal{H}(\varphi|Y = \bar{y}) = \Sigma_{i=1}^{k}\left(\frac{\mathsf{cnt}_{YB}(\bar{y}, b_i)}{|\Delta(\bar{y})|} * \log_k \frac{|\Delta(\bar{y})|}{\mathsf{cnt}_{YB}(\bar{y}, b_i)}\right),$$

where (a) $k = |\pi_B(\Delta(\bar{y}))|$, the number of distinct B values in $\Delta(\bar{y})$, (b) for each $i \in [1, k]$, $b_i \in \pi_B(\Delta(\bar{y}))$, (c) $\mathsf{cnt}_{YB}(\bar{y}, b_i)$ denotes the number of tuples $t \in \Delta(\bar{y})$ with $t[B] = b_i$, and (d) $|\Delta(\bar{y})|$ is the number of tuples in $\Delta(\bar{y})$.

Intuitively, we treat $\mathcal{X}(\varphi|Y = \bar{y})$ as a random variable for the value of the B attribute in $\Delta(\bar{y})$, with a set $\pi_B(\Delta(\bar{y}))$ of possible values. The probability for b_i to be the value is $p_i = \frac{\mathsf{cnt}_{YB}(\bar{y}, b_i)}{|\Delta(\bar{y})|}$. Then when $\mathcal{H}(\varphi|Y = \bar{y})$ is small enough, it is highly accurate to resolve the conflict by letting $t[B] = b_j$ for each $t \in \Delta(\bar{y})$, where b_j is the one with the highest probability, or in other words, when $\mathsf{cnt}_{YB}(\bar{y}, b_j)$ is the maximum among all $b_i \in \pi_B(\Delta(\bar{y}))$.

In particular, $\mathcal{H}(\varphi|Y = \bar{y}) = 1$ when $\mathsf{cnt}_{YB}(\bar{y}, b_i) = \mathsf{cnt}_{BA}(\bar{y}, b_j)$ for all distinct $b_i, b_j \in \pi_B(\Delta(\bar{y}))$, and if $\mathcal{H}(\varphi|Y = \bar{y}) = 0$ for all $\bar{y} \in \pi_Y(\rho_{Y \asymp t_p[Y]} D)$, then $D \models \varphi$.

6.2 Entropy-based Data Cleaning

We first describe an algorithm based on entropy, followed by presenting its main procedures and auxiliary structures.

Algorithm. The algorithm, referred to as eRepair, is shown in Fig. 4. Given a set Σ of CFDs, a set Γ of MDs, a master relation D_m, dirty data D, and two thresholds δ_1 and δ_2 for *update frequency* and *entropy*, respectively, it finds reliable fixes for D and returns a (partially cleaned) database D'

Algorithm eRepair

Input: CFDs Σ, MDs Γ, master data D_m, dirty data D,
update threshold δ_1, entropy threshold δ_2.
Output: A partial repair D' of D with reliable fixes.

```
1.  O := the order of Σ ∪ Γ, sorted via their dependency graph;
2.  D' := D;
3.  repeat
4.      for (i = 1; i ≤ |Σ ∪ Γ|; i++) do
5.          ξ := the i-th rule in O;
6.          case ξ of
7.          (1) variable CFD: D' := vCFDReslove(D', ξ, δ1, δ2);
8.          (2) constant CFD: D' := cCFDReslove(D', ξ, δ1);
9.          (3) MD:           D' := MDReslove(D', Dm, ξ, δ1);
10. until there are no changes in D'
11. return D'.
```

Figure 4: Algorithm eRepair

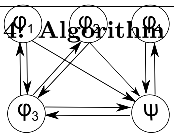


Figure 5: Example dependency graph

in which reliable fixes are marked. The deterministic fixes found earlier by cRepair remain unchanged in the process.

The algorithm first finds an order $\mathcal{O}$ on the rules in $\Sigma \cup \Gamma$ (line 1). It then repeatedly applies the rules in the order $\mathcal{O}$ to resolve conflicts in D (lines 3–10), by invoking procedures vCFDReslove (line 7), cCFDReslove (line 8) or MDReslove (line 9), based on the types of the rules (lines 5-6). It terminates when either no more rules can be applied or all data values have been changed more than δ_1 times, *i.e.,* when there is no enough information to make reliable fixes (line 10). A partially cleaned database is returned with reliable fixes being marked (line 11). In a nutshell, algorithm eRepair first sorts cleaning rules derived from the CFDs and MDs, such that rules with relatively bigger impact are applied early. Following the order, it then applies the rules one by one, until no more reliable fixes can be found.

Procedures. We next present those procedures.

Sorting cleaning rules. To avoid unnecessary computation, we sort $\Sigma \cup \Gamma$ based on its *dependency graph* $G = (V, E)$. Each rule of $\Sigma \cup \Gamma$ is a node in V, and there is an edge from a rule ξ_1 to another ξ_2 if ξ_2 can be applied after the application of ξ_1. There exists an edge $(u, v) \in E$ from node u to node v if $\text{RHS}(\xi_u) \cap \text{LHS}(\xi_v) \neq \emptyset$. Intuitively, edge (u, v) indicates that whether ξ_v can be applied depends on the outcome of applying ξ_u. Hence, ξ_u is applied before ξ_v. For instance, the dependency graph of the CFDs and MDs given in Example 1.1 is shown in Fig. 5.

Based on G, we sort the rules as follows. (1) Find strongly connected components (SCCs) in G, in linear time [11]. (2) By treating each SCC as a single node, we convert G into a DAG. (3) Find a topological order on the nodes in the DAG. That is, a rule ξ_1 is applied before another ξ_2 if the application of ξ_1 affects the application of ξ_2. (4) Finally, the nodes in each SCC are further sorted based on the ratio of its out-degree to in-degree, in a decreasing order. The higher the ratio is, the more effects it has on other nodes.

Example 6.1: The dependency graph G in Fig. 5 is an SCC. The ratios of out-degree to in-degree of the nodes φ_1, φ_2, φ_3, φ_4 and ψ are $\frac{2}{1}$, $\frac{2}{1}$, $\frac{1}{1}$, $\frac{3}{3}$ and $\frac{2}{4}$, respectively. Hence the order $\mathcal{O}$ of these rules is $\varphi_1 > \varphi_2 > \varphi_3 > \varphi_4 > \psi$, where those nodes with the same ratio are sorted randomly. □

vCFDReslove. It applies the cleaning rule derived from a variable CFD $\xi = R(Y \to B, t_p)$. For each set $\Delta(\bar{y})$ with $\bar{y}$ in $\pi_Y(\rho_{Y \asymp t_p[Y]} D)$, if $\mathcal{H}(\xi | Y = \bar{y})$ is smaller than the entropy threshold δ_2, it picks the value $b \in \pi_B(\Delta(\bar{y}))$ that has the maximum $\mathsf{cnt}_{YB}(\bar{y}, b)$. Then for each tuple $t \in \Delta(\bar{y})$, if $t[B]$ has been changed less than δ_1 times, *i.e.,* when $t[B]$ is not often changed by rules that may not converge on its value, $t[B]$ is changed to b. As remarked earlier, when the entropy $\mathcal{H}(\xi | Y = \bar{y})$ is small enough, it is highly accurate to resolve the conflicts in $\pi_B(\Delta(\bar{y}))$ by assigning b as their value.

	A	B	C	E	F	H
t_1:	a_1	b_1	c_1	e_1	f_1	h_1
t_2:	a_1	b_1	c_1	e_1	f_2	h_2
t_3:	a_1	b_1	c_1	e_1	f_3	h_3
t_4:	a_1	b_1	c_1	e_2	f_1	h_3
t_5:	a_2	b_2	c_2	e_1	f_2	h_4
t_6:	a_2	b_2	c_2	e_2	f_1	h_4
t_7:	a_2	b_2	c_3	e_3	f_3	h_5
t_8:	a_2	b_2	c_4	e_3	f_3	h_6

Figure 6: Example relation of schema R

cCFDReslove. It applies the rule derived from a constant CFD $\xi = R(X \to A, t_{p_1})$. For each tuple $t \in D$, if (a) $t[X] \asymp t_{p_1}[X]$, (b) $t[A] \neq t_{p_1}[A]$, and (c) $t[A]$ has been changed less than δ_1 times, then $t[A]$ is changed to the constant $t_{p_1}[A]$.

MDReslove. It applies the rule derived from an MD $\xi = \bigwedge_{j \in [1,k]} (R[A_j] \approx_j R_m[B_j]) \to R[E] \rightleftharpoons R_m[F]$. For each tuple $t \in D$, if there exists a master tuple $s \in D_m$ such that (a) $t[A_j] \approx_j s[B_j]$ for $j \in [1, k]$, (b) $t[E] \neq s[F]$, and (c) $t[E]$ has been changed less than δ_1 times, then it assigns the master value $s[F]$ to $t[E]$.

These procedures do not change those data values that are marked deterministic fixes by algorithm cRepair.

We next show by example how algorithm cRepair works.

Example 6.2: Consider an instance of schema $R(\mathsf{ABCEFH})$ shown in Fig. 6, and a variable CFD $\phi = R(\mathsf{ABC} \to \mathsf{E}, t_{p_1})$, where t_{p_1} consists of wildcards only, *i.e.,* ϕ is an FD. Observe that (a) $\mathcal{H}(\phi | ABC = (a_1, b_1, c_1)) \approx 0.8$ (b) $\mathcal{H}(\phi | ABC = (a_2, b_2, c_2))$ is 1, and (c) $\mathcal{H}(\phi | ABC = (a_2, b_2, c_3))$ and $\mathcal{H}(\phi | ABC = (a_2, b_2, c_4))$ are both 0.

From these we can see the following. (1) For $\Delta(ABC = (a_2, b_2, c_3))$ and $\Delta(ABC = (a_2, b_2, c_4))$, the entropy is 0; hence these sets of tuples do not violate ϕ, *i.e.,* there is no need to fix these tuples. (2) The fix based on $\mathcal{H}(\phi | ABC = (a_1, b_1, c_1))$ is relatively accurate, but not those based on $\mathcal{H}(\phi | ABC = (a_2, b_2, c_2))$. Hence the algorithm will only change $t_4[E]$ to e_1, and marks it as a reliable fix.

In contrast, the data D of Fig. 1 has too few tuples to infer sensible entropy. No reliable fixes are found for D. □

Complexity. The outer loop (lines 3–10) in algorithm eRepair runs in $O(\delta_1 |D|)$ time. Each inner loop (lines 4–9) takes $O(|D||\Sigma| + k|D|\mathsf{size}(\Gamma))$ time using the optimization techniques of Section 5.1, where k is a constant. Thus, the algorithm takes $O(\delta_1 |D|^2 |\Sigma| + \delta_1 k |D|^2 \mathsf{size}(\Gamma))$ time.

6.3 Resolving Conflicts with a 2-in-1 Structure

We can efficiently identify tuples that match the LHS of constant CFDs by building an index on the LHS attributes in the database D. We can also efficiently find tuples that match the LHS of MDs by leveraging the suffix tree structure developed in Section 5. However, for variable CFDs, two issues still remain: (a) detecting violations and (b) computing entropy. These are rather costly and have to be recomputed when data is updated in the cleaning process. To do these we develop a 2-in-1 structure, which can be easily maintained.

Let Σ_V be the set of variables CFDs in Σ, and $\mathsf{attr}(\Sigma_V)$ be the set of attributes appearing in Σ_V. For each CFD φ

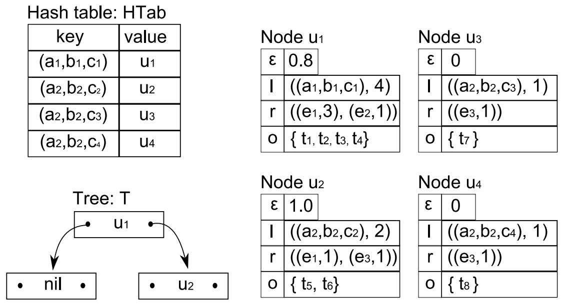


Figure 7: Example data structure for variable CFDs

$= R(Y \to B,\ t_p)$ in Σ_V, we build a structure consisting of a *hash table* and an *AVL tree* [11] T as follows.

Hash table HTab. Recall $\Delta(\bar{y}) = \{t \mid t \in D,\ t[Y] = \bar{y}\}$ for $\bar{y} \in \pi_Y(\rho_{Y \asymp t_p[Y]} D)$ described earlier. For each $\Delta(\bar{y})$, we insert an entry (key, val) into HTab, where key $= \bar{y}$, and val is a pointer linking to a node $\mathsf{u} = (\epsilon, l, r, o)$, where (a) $\mathsf{u}.\epsilon = \mathcal{H}(\varphi|Y = \bar{y})$, (b) $\mathsf{u}.l$ is the value-count pair $(\bar{y}, |\Delta(\bar{y})|)$, (c) $\mathsf{u}.r$ is the set $\{(b, \mathsf{cnt}_{YB}(\bar{y}, b)) \mid b \in \pi_B(\Delta(\bar{y}))\}$, and (d) $\mathsf{u}.o$ is the set of (partial) tuple IDs $\{t.id \mid t \in \Delta(\bar{y})\}$.

AVL tree T. For each $\bar{y} \in \pi_Y(\rho_{Y \asymp t_p[Y]} D)$ with entropy $\mathcal{H}(\varphi|Y = \bar{y}) \neq 0$, we create a node $\mathsf{v} = \mathsf{HTab}(\bar{y})$ in T, a pointer to the node u for $\Delta(\bar{y})$ in HTab. For each node v in T, its left child $\mathsf{v}_l.\epsilon \leq \mathsf{v}.\epsilon$ and its right child $\mathsf{v}_r.\epsilon \geq \mathsf{v}.\epsilon$.

Note that both the number HTab of entries in the hash table HTab and the number $|T|$ of nodes in the AVL tree T are bounded by the number $|D|$ of tuples in D.

Example 6.3: Consider the relation in Fig. 6 and the variable CFD ϕ given in Example 6.2. The hash table HTab and the AVL tree T for ϕ are shown in Fig. 7. □

We next show how to use and maintain the structures.

(1) Lookup cost. For the CFD φ, it takes (a) $O(\log |T|)$ time to identify the set $\Delta(\bar{y})$ of tuples with minimum entropy $\mathcal{H}(\varphi|Y = \bar{y})$ in the AVL tree T, and (b) $O(1)$ time to check whether two tuples in D satisfy φ via the hash table HTab.

(2) Update cost. The initialization of both the hash table HTab and the AVL tree T can be done by scanning the database D once, and it takes $O(|D| \log |D| |\Sigma_V|)$ time.

After resolving some conflicts, the structures need to be maintained accordingly. Consider a set $\Delta(\bar{y})$ of dirty tuples. When a reliable fix is found for $\Delta(\bar{y})$ based on $\mathcal{H}(\varphi|Y = \bar{y})$, we do the following: (a) remove a node from tree T, which takes $O(\log |T|)$ time, where $|T| \leq |D|$; and (b) update the hash tables and trees for all other CFDs, which takes $O(|\Delta(\bar{y})||\Sigma_V| + |\Delta(\bar{y})| \log |D|)$ time in total.

(3) Space cost. The structures take $O(|D|\mathsf{size}(\Sigma_V)$ space for all CFDs in Σ_V in total, where $\mathsf{size}(\Sigma_V)$ is the size of Σ_V.

Putting these together, the structures are efficient in both time and space, and are easy to be maintained.

7. Experimental Study

We next present an experimental study of our data cleaning techniques underlying UniClean, which unify matching and repairing operations. Using real-life data, we evaluated (1) the effectiveness of our data cleaning algorithms, (2) the accuracy of deterministic fixes and reliable fixes, and (3) the scalability of our algorithms with the size of data.

Experimental Setting. We used two real-life data sets.

(1) HOSP *data* was taken from US Department of Health & Human Services[1]. It has 100K records with 19 attributes. We designed 23 CFDs and 3 MDs for HOSP, 26 in total.

(2) DBLP *data* was extracted from DBLP Bibliography[2]. It consists of 400K tuples, each with 12 attributes. We designed 7 CFDs and 3 MDs for DBLP, 10 in total.

(3) Master data for both datasets was carefully selected from the same data sources so that they were guaranteed to be correct and consistent *w.r.t.* the designed rules.

(4) Dirty datasets were produced by introducing noises to data from the two sources, controlled by four parameters: (a) $|D|$: the data size; (b) noi%: the *noise rate*, which is the ratio of the number of erroneous attributes to the total number of attributes in D; (c) dup%: the *duplicate rate*, *i.e.,*, the percentage of tuples in D that can find a match in the master data; and (d) asr%: the *asserted rate*. For each attribute A, we randomly picked asr% of tuples t from the data and set $t[A].\mathsf{cf} = 1$, while letting $t'[A].\mathsf{cf} = 0$ for the other tuples t'. The default value for asr% is 40%.

Algorithms. We implemented the following algorithms, all in Python: (a) algorithms cRepair, eRepair and hRepair (an extension of algorithm in [10]) in UniClean; (b) the sorted neighborhood method of [25], denoted by SortN, for record matching based on MDs only; and (c) the heuristic repairing algorithm of [10], denoted by quaid, based on CFDs only. We use Uni to denote cleaning based on both CFDs and MDs (matching and repairing), and Uni(CFD) to denote cleaning using CFDs (repairing) only.

We used edit distance for similarity test, defined as the minimum number of single-character insertions, deletions and substitutions needed to convert a value from v to v'.

Quality measuring. We adopted *precision, recall* and *F-measure*, which are commonly used in information retrieval, where F-measure $= 2 \cdot (\text{precision} \cdot \text{recall})/(\text{precision} + \text{recall})$.

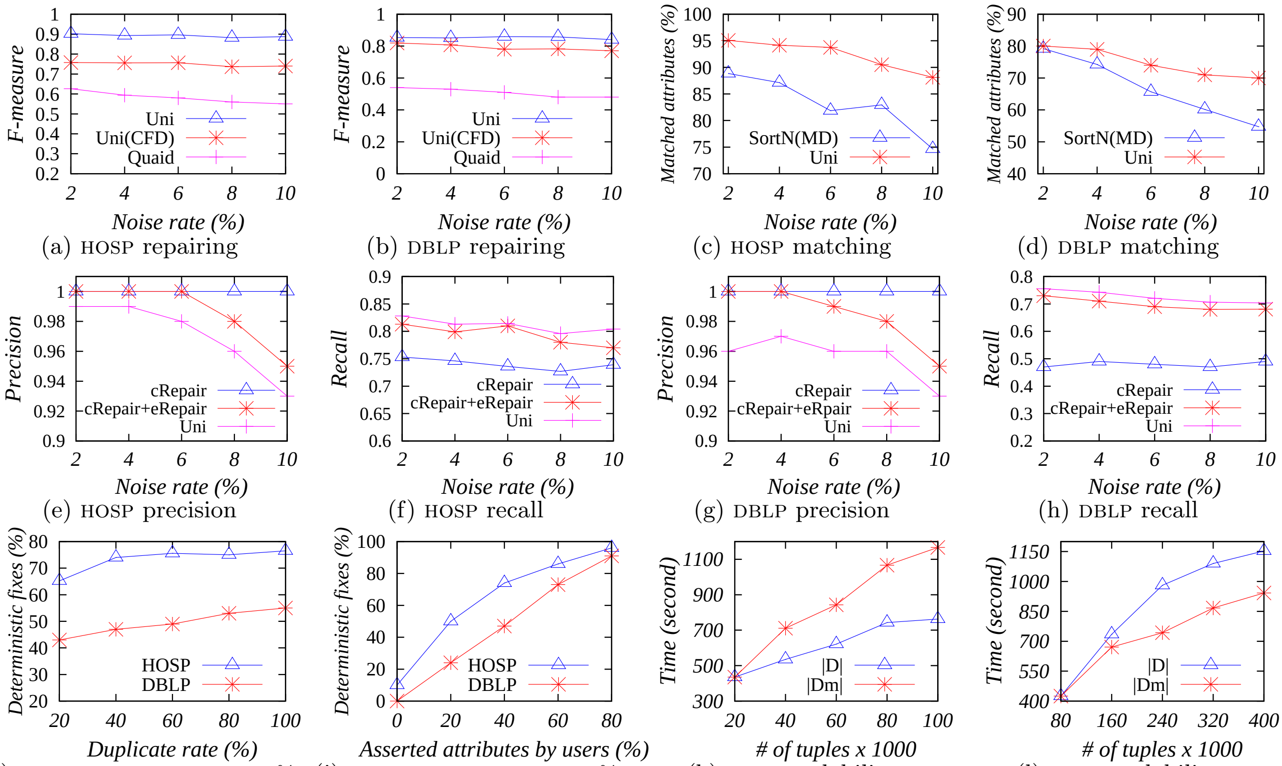
For record matching, (a) precision is the ratio of *true matches* (true positives) correctly found by an algorithm to all the duplicates found, and (b) recall is the ratio of true matches correctly found to all the matches between a dataset and master data. For data repairing, (a) precision is the ratio of attributes correctly updated to the number of all the attributes updated, and (b) recall is the ratio of attributes corrected to the number of all erroneous attributes.

All experiments were conducted on a Linux machine with a 3.0GHz Intel CPU and 4GB of Memory. Each experiment was run more than 5 times, and the average is reported here.

Experimental Results. We conducted five sets of experiments: (a) in the first two sets of experiments, we compared the effectiveness of our cleaning methods with both matching and repairing against its counterpart with only matching or only repairing; (b) we evaluated the accuracy of deterministic fixes, reliable fixes and possible fixes in the third set of experiments; (c) we evaluated the impact of the duplicate rate and asserted rate on the percentage of deterministic fixes found by our algorithm cRepair in the fourth set of experiments; and (d) the last set of experiments tested the scalability of Uni with both the size of dirty data and the size of master data. In all the experiments, we set the threshold for entropy and confidence to be 0.8 and 1.0, respectively.

[1]http://www.hospitalcompare.hhs.gov/

[2]http://www.informatik.uni-trier.de/~ley/db/



(a) HOSP repairing (b) DBLP repairing (c) HOSP matching (d) DBLP matching

(e) HOSP precision (f) HOSP recall (g) DBLP precision (h) DBLP recall

(i) Deterministic fixes on dup% (j) Deterministic fixes on asr% (k) HOSP scalability (l) DBLP scalability

Figure 8: Experimental results

We used dirty datasets and master data consisting of 60K tuples each. We now report our findings.

Exp-1: Matching helps repairing. In the first set of experiments we show that matching indeed helps repairing. We compare the quality (F-measure) of fixes generated by Uni, Uni(CFD) and quaid. Fixing the duplicate rate dup% = 40%, we varied the noise rate noi% from 2% to 10%. Observe that dup% is only related to matching via MDs. To favor Uni(CFD) and quaid, which use CFDs only, we focused on the impact of various noise rates.

The results on HOSP data and DBLP data are reported in Figures 8(a) and 8(b), respectively, which tell us the following. (1) Uni clearly outperforms Uni(CFD) and quaid by up to 15% and 30%, respectively. This verifies that matching indeed helps repairing. (2) The F-measure decreases when noi% increases for all three approaches. However, Uni with matching is less sensitive to noi%, which is another benefit of unifying repairing with matching. (3) Even only with CFDs, our system Uni(CFD) still outperforms quaid, as expected. This is because quaid only generates possible fixes with heuristic, while Uni(CFD) finds both deterministic fixes and reliable fixes. This also verifies that deterministic and reliable fixes are more accurate than possible fixes.

Exp-2: Repairing helps matching. In the second set of experiment, we show that repairing indeed helps matching. We evaluate the quality (F-measure) of matches found by (a) Uni and (b) SortN using MDs, denoted by SortN(MD). We used the same setting as in Exp-1. We also conducted experiments by varying the duplicate rate, but found that its impact is very small; hence we do not report it here.

The results are reported in Figures 8(c) and 8(d) for HOSP data and DBLP data, respectively. We find the following. (a) Uni outperforms SortN(MD) by up to 15%, verifying that repairing indeed helps matching. (b) The F-measure decreases when the noise rate increases for both approaches. However, Uni with repairing is less sensitive to noi%, which is consistent with our observation in the last experiments.

Exp-3: Accuracy of deterministic and reliable fixes. In this set of experiments we evaluate the accuracy (precision and recall) of (a) deterministic fixes generated in the first phase of UniClean, denoted by cRepair, (b) deterministic fixes and reliable fixes generated in the first two phases of UniClean, denoted by cRepair + eRepair, and (c) all fixes generated by Uni. Fixing dup% = 40%, we varied noi% from 2% to 10%. The results are reported in Figures 8(e)–8(h).

The results tell us the following: (a) Deterministic fixes have the highest precision, and are insensitive to the noise rate. However, their recall is low, since cRepair is "picky": it only generates fixes with asserted attributes. (b) Fixes generated by Uni have the lowest precision, but the highest recall, as expected. Further, their precision is quite sensitive to noi%. This is because the last step of UniClean is by heuristics, which generates possible fixes. (c) The precision and recall of deterministic fixes and reliable fixes by cRepair + eRepair are in the between, as expected. Further, their precision is also sensitive to noi%. From these we can see that the precision of reliable fixes and possible fixes is sensitive to noi%, but not their recall. Moreover, when noi% is less than 4%, their precision is rather indifferent to noi%.

Exp-4: Impact of dup% and asr% on deterministic fixes. In this set of experiments we evaluated the percentage of deterministic fixes found by algorithm cRepair.

Fixing the asserted rate asr% = 40%, we varied the duplicate rate dup% from 20% to 100%. Figure 8(i) shows the results. We find that the larger dup% is, the more deterministic fixes are found, as expected.

Fixing dup% = 40%, we varied asr% from 0% to 80%. The results are shown in Fig. 8(j), which tell us that the number of deterministic fixes found by cRepair highly depends on asr%. This is because to find deterministic fixes, cleaning rules are only applied to asserted attributes.

Exp-5: Scalability. The last experiments evaluated the scalability of Uni with the size $|D|$ of dirty data and the size $|D_m|$ of master data. We fixed noi% = 6% and dup% = 40% in these experiments. The results are reported in Figures 8(k) and 8(l) for HOSP and DBLP data, respectively.

Figure 8(k) shows two curves for HOSP data: one by fixing $|D_m|$ = 60K and varying $|D|$ from $20K$ to $100K$, and the other by fixing $|D|$ = 60K and varying $|D_m|$ from $20K$ to $100K$. The results show that Uni scales reasonably well with both $|D|$ and $|D_m|$. In fact Uni scales much better than quaid [10]: quaid took more than 10 hours when $|D|$ is 80K, while it took Uni about 11 minutes. These results verify the effectiveness of our indexing structures and optimization techniques developed for Uni. The results are consistent for DBLP data, as shown in Fig. 8(l).

Summary. From the experimental results on real-life data, we find the following. (a) Data cleaning by unifying matching and repairing operations substantially improves the quality of fixes: it outperforms matching and repairing taken as independent processes by up to 30% and 15%, respectively. (b) Deterministic fixes and reliable fixes are highly accurate. For example, when the noise rate is no more than 4%, their precision is close to 100%. The precision decreases slowly when increasing noise rate. These tell us that it is feasible to find accurate fixes for real-life applications. (c) Candidate repairs generated by system UniClean are of high-quality: their precision is about 96%. (d) Our data cleaning methods scale reasonably well with the size of data and the size of master data. It performs better than quaid, a data repairing tool using CFDs only. Indeed, it is more than 50 times faster than quaid.

8. Conclusion

We have taken a first step toward unifying record matching and data repairing, an important issue that has been overlooked by and large. We have proposed a uniform framework for interleaving matching and repairing operations, based on cleaning rules derived from CFDs and MDs. We have established the complexity bounds of several fundamental problems for data cleaning with both matching and repairing. We have also proposed deterministic fixes and reliable fixes, and effective methods to find these fixes based on confidence and entropy. Our experimental results have verified that our techniques substantially improve the quality of fixes generated by repairing and matching taken separately.

We are currently experimenting with larger datasets and exploring optimization techniques to improve the efficiency of our algorithms. We are also studying cleaning of multiple relations of which the consistency is specified by constraints across relations, *e.g.,* (conditional) inclusion dependencies.

Acknowledgments. Fan is supported in part by the RSE-NSFC Joint Project Scheme and an IBM scalable data analytics for a smarter planet innovation award. Li is supported in part by NGFR 973 grant 2006CB303000 and NSFC grant 60533110. Shuai is supported in part by NGFR 973 grant 2011CB302602 and NSFC grants 90818028 and 60903149.

9. References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[2] A. Aiken, D. Kozen, M. Y. Vardi, and E. L. Wimmers. The complexity of set constraints. In *CSL*, 1993.

[3] A. Arasu, C. Re, and D. Suciu. Large-scale deduplication with constraints using Dedupalog. In *ICDE*, 2009.

[4] M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. *TPLP*, 3(4-5), 2003.

[5] C. Batini and M. Scannapieco. *Data Quality: Concepts, Methodologies and Techniques.* Springer, 2006.

[6] G. Beskales, M. A. Soliman, I. F. Ilyas, and S. Ben-David. Modeling and querying possible repairs in duplicate detection. In *VLDB*, 2009.

[7] P. Bohannon, W. Fan, M. Flaster, and R. Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *SIGMOD*, 2005.

[8] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. Robust and efficient fuzzy match for online data cleaning. In *SIGMOD*, 2003.

[9] F. Chiang and R. Miller. Discovering data quality rules. In *VLDB*, 2008.

[10] G. Cong, W. Fan, F. Geerts, X. Jia, and S. Ma. Improving data quality: Consistency and accuracy. In *VLDB*, 2007.

[11] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* The MIT Press, 2001.

[12] T. M. Cover and J. A. Thomas. *Elements of Information Theory.* Wiley-Interscience, 1991.

[13] T. de Vries, H. Ke, S. Chawla, and P. Christen. Robust record linkage blocking using suffix arrays. In *CIKM*, 2009.

[14] X. Dong, A. Y. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *SIGMOD Conference*, 2005.

[15] X. L. Dong, L. Berti-Equille, Y. Hu, and D. Srivastava. Global detection of complex copying relationships between sources. In *VLDB*, 2010.

[16] W. W. Eckerson. Data Quality and the Bottom Line: Achieving Business Success through a Commitment to High Quality Data. In *The Data Warehousing Institute*, 2002.

[17] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *TKDE*, 19(1):1–16, 2007.

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

[19] W. Fan, X. Jia, J. Li, and S. Ma. Reasoning about record matching rules. In *VLDB*, 2009.

[20] W. Fan, J. Li, S. Ma, N. Tang, and W. Yu. Towards certain fixes with editing rules and master data. In *VLDB*, 2010.

[21] I. Fellegi and D. Holt. A systematic approach to automatic edit and imputation. *J. American Statistical Association*, 71(353):17–35, 1976.

[22] Gartner. Forecast: Data quality tools, worldwide, 2006-2011. Technical report, Gartner, 2007.

[23] S. Guo, X. Dong, D. Srivastava, and R. Zajac. Record linkage with uniqueness constraints and erroneous values. *PVLDB*, 3(1), 2010.

[24] R. W. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 29(2):147–160, 1950.

[25] M. A. Hernandez and S. Stolfo. Real-World Data is Dirty: Data Cleansing and the Merge/Purge Problem. *Data Mining and Knowledge Discovery*, 2(1):9–37, 1998.

[26] T. N. Herzog, F. J. Scheuren, and W. E. Winkler. *Data Quality and Record Linkage Techniques.* Springer, 2009.

[27] G. J. Klir and T. A. Folger. *Fuzzy sets, uncertainty, and information.* Englewood Cliffs, N.J: Prentice Hall, 1988.

[28] D. Loshin. *Master Data Management.* Knowledge Integrity, Inc., 2009.

[29] C. Mayfield, J. Neville, and S. Prabhakar. ERACER: a database approach for statistical inference and data cleaning. In *SIGMOD*, 2010.

[30] F. Naumann, A. Bilke, J. Bleiholder, and M. Weis. Data fusion in three steps: Resolving schema, tuple, and value inconsistencies. *IEEE Data Eng. Bull.*, 29(2), 2006.

[31] B. Otto and K. Weber. From health checks to the seven sisters: The data quality journey at BT, Sept. 2009. BT TR-BE HSG/CC CDQ/8.

[32] T. Redman. The impact of poor data quality on the typical enterprise. *Commun. ACM*, 2:79–82, 1998.

[33] S. Song and L. Chen. Discovering matching dependencies. In *CIKM*, 2009.

[34] D. Srivastava and S. Venkatasubramanian. Information theory for data management. In *SIGMOD*, 2010.

[35] I. Wegener and R. Pruim. *Complexity Theory: Exploring the Limits of Efficient Algorithms*. Springer, 2005.

[36] M. Weis and F. Naumann. Dogmatix tracks down duplicates in xml. In *SIGMOD*, 2005.

[37] S. E. Whang, O. Benjelloun, and H. Garcia-Molina. Generic entity resolution with negative rules. *VLDB J.*, 18(6), 2009.

[38] J. Wijsen. Database repairing using updates. *TODS*, 30(3):722–768, 2005.

[39] M. Yakout, A. K. Elmagarmid, J. Neville, and M. Ouzzani. GDR: a system for guided data repair. In *SIGMOD*, 2010.

[40] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE TIT*, 24(5), 1978.