A

Interaction between Record Matching and Data Repairing

Wenfei Fan, University of Edinburgh, and SKLSDE Lab, Beihang University
Shuai Ma, SKLSDE Lab, Beihang University
Nan Tang, QCRI
Wenyuan Yu, University of Edinburgh

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 integrity constraints. These are typically treated as separate processes in current data cleaning systems, based on heuristic solutions. This paper studies a new problem in connection with data cleaning, 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 provide 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-complete 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 analyses, respectively, which are more accurate than fixes generated by heuristics. Heuristic fixes are produced only when deterministic or reliable fixes are unavailable. We experimentally verify that our techniques can significantly improve the accuracy of record matching and data repairing that are taken as separate processes, using real-life and synthetic data.

Categories and Subject Descriptors: H.2.m [**Database Management**]: Miscellaneous—*Data cleaning*

General Terms: Design, Algorithm, Performance

Additional Key Words and Phrases: Data repairing, Record matching, Conditional functional dependency, Matching dependency

1. INTRODUCTION

It has long been recognized that data residing in a database is often dirty [Redman 1998]. Dirty data inflicts a daunting cost: it costs US businesses 600 billion dollars each year [Eckerson 2002]. 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 [Otto and Weber 2009]. 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 [Gartner 2007].

There are two central issues about data cleaning:

- *Recording matching* is to identify tuples that refer to the same real-world entity [Elmagarmid et al. 2007; Herzog et al. 2009].
- *Data repairing* is to find another database (a candidate repair) that is consistent and minimally differs from the original data, by detecting and fixing errors in the data [Arenas et al. 2003; Fellegi and Holt 1976].

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 by the example 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*

	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 28/08/10	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 28/09/10	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 06/11/09	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	ring, 2,100 USD	1pm 06/11/09	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

Fig. 1. Example master data and database

master data [Loshin 2009]; and D consists of transaction records of credit cards, which may be dirty. The databases D_m and D are specified by schemas card and tran given below, respectively:

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

Here a card tuple specifies a UK credit card holder identified by his 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 at time when, by a UK customer who is identified by his name (FN, LN), address (St, city, post code), area code (AC), phone (phn) and gender (gd). Example instances of both card and tran relations are shown in Figures. 1(a) and 1(b), which are fractions of master data D_m and transaction records D, respectively (the cf rows in Fig. 1(b) will be discussed later).

Following [Fan et al. 2008; Fan et al. 2011a], we use conditional functional dependencies (CFDs [Fan et al. 2008]) φ_1–φ_4 to specify the consistency of the tran data in D, and we employ a matching dependency (MD [Fan et al. 2011a]) ψ as a rule for matching tuples across D and master card data D_m:

φ_1: $\mathsf{tran}([\mathsf{AC} = 131] \rightarrow [\mathsf{city} = \mathrm{Edi}])$,
φ_2: $\mathsf{tran}([\mathsf{AC} = 020] \rightarrow [\mathsf{city} = \mathrm{Ldn}])$,
φ_3: $\mathsf{tran}([\mathsf{city}, \mathsf{phn}] \rightarrow [\mathsf{St}, \mathsf{AC}, \mathsf{post}])$,
φ_4: $\mathsf{tran}([\mathsf{FN} = \mathrm{Bob}] \rightarrow [\mathsf{FN} = \mathrm{Robert}])$,
ψ: $\mathsf{tran}[\mathsf{LN}, \mathsf{city}, \mathsf{St}, \mathsf{post}] = \mathsf{card}[\mathsf{LN}, \mathsf{city}, \mathsf{St}, \mathsf{zip}] \wedge \mathsf{tran}[\mathsf{FN}] \approx \mathsf{card}[\mathsf{FN}]$
$\rightarrow \mathsf{tran}[\mathsf{FN}, \mathsf{phn}] \rightleftharpoons \mathsf{card}[\mathsf{FN}, \mathsf{tel}]$,

where (1) CFD φ_1 (resp. φ_2) asserts that, for any tran tuple, if its 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 and postal code; (3) CFD φ_4 is a data standardization rule: if the first name is Bob, then it should be "normalized" to be Robert; and (4) 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 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 (counting 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 tuples 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 CFD φ_2, and t'_3[FN] = Robert by normalization with φ_4;
(b) match t'_3 with master tuple 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]; and
(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 assured 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 that they indeed refer to the same person; hence a fraud.

Observe that not only repairing helps matching (*e.g.,* from step (a) to step (b)), but matching also helps us repair the data (*e.g.,* step (d) is doable only after the matching operation in step (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 run these processes consecutively one after another. There has been a host of work on record matching (*e.g.,* [Arasu et al. 2009; Beskales et al. 2009; Chaudhuri et al. 2003; Fan et al. 2011a; Hernandez and Stolfo 1998; Whang et al. 2009]; see [Elmagarmid et al. 2007; Herzog et al. 2009] for surveys) as well as on data repairing (*e.g.,* [Arenas et al. 2003; Bohannon et al. 2005; Cong et al. 2007; Fan et al. 2010; Fellegi and Holt 1976; Mayfield et al. 2010; Yakout et al. 2010]). However, the problem of unifying record matching and data repairing to improve the accuracy of data has not been well addressed.

Contributions. We make a first effort to clean data by *unifying* record matching and data repairing, and to provide a data cleaning solution that stresses the accuracy.

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

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

As opposed to record matching or 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 [Loshin 2009]. It is being widely used in industry, supported by many corporations, *e.g.,* IBM, SAP, Microsoft and Oracle. To identify the tuples from D and D_m, we use matching rules that are an extension of MDs [Fan et al. 2011a] by supporting negative rules (*e.g.,* a male and a female may not refer to the same person) [Arasu et al. 2009; Whang et al. 2009].

(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* values 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 [Cover and Thomas 1991; Srivastava and Venkatasubramanian 2010], and (c) master data [Loshin 2009]. We distinguish three classes of fixes: (i) *deterministic* fixes for corrections derived from attributes that are asserted correct based on the confidence; (ii) *reliable* fixes for those derived using entropy; and (iii) *possible* fixes for those generated by heuristics. The former two classes are more accurate than the last class of fixes, *i.e.,* 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 [Fan et al. 2008], *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 [Wegener and Pruim 2005] unless P = NP. (c) It is more challenging to decide whether a data cleaning process can terminate and whether different cleaning processes yield the same 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 the confidence analysis and master data. (b) When confidence is low or unavailable, we provide another algorithm to compute *reliable fixes* by employing information entropy, inferring evidence from data itself to improve the accuracy. (c) To fix the remaining errors, we extend the heuristic based method [Cong et al. 2007] 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), as well as synthetic data (TPC-H). We find that our methods substantially outperform matching and repairing taken as separate processes in the accuracy of fixes. 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 important and feasible in practice, and that it should logically become part of data cleaning systems. While master data is desirable in the process, *it is not a must*. When master data is absent, although deterministic fixes may have lower accuracy, reliable and heuristic fixes would not degrade substantially. Indeed, in its absence, our approach can be adapted by interleaving (a) record matching in a single data table with MDs, as described in [Fan et al. 2011a], and (b) data repairing with CFDs.

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. Algorithms for finding deterministic and reliable fixes are provided in Sections 5 and 6, respectively. Section 7 discusses possible fixes with heuristics. Section 8 reports experimental study. Section 9 identifies open issues.

Related work. This work extends [Fan et al. 2011c] by including (1) proofs of all the fundamental problems in connection with data cleaning (Sections 3 and 4); (2) a detailed analysis of optimization techniques for deterministic fixes (Section 5.2); (3)

analysis of possible fixes based on heuristics (Section 7); and (4) new experimental study using synthetic data for scalability. None of these was given in [Fan et al. 2011c]. Some of the proofs are nontrivial and are interesting in their own right.

Record matching is also known as record linkage, entity resolution, merge-purge and duplicate detection [Arasu et al. 2009; Beskales et al. 2009; Chaudhuri et al. 2003; Fan et al. 2011a; Hernandez and Stolfo 1998; Whang et al. 2009; Weis and Naumann 2005; Dong et al. 2005; Guo et al. 2010; Cao et al. 2011; Christen 2012], and object identification (see [Herzog et al. 2009; Elmagarmid et al. 2007] for surveys), among which a line of these work focus on the scalability issue, *e.g.,* [Hernandez and Stolfo 1998; Cao et al. 2011; Christen 2012]. There has also been work on speeding-up the computation of particular similarity metrics, *e.g.,* edit distance [Wang et al. 2010; Xiao et al. 2008], Jaccard similarity [Xiao et al. 2011] and fuzzy token matching based similarity [Wang et al. 2011], and on the similarity computation of a collection of sparse vectors [Bayardo et al. 2007] as well. Some of the techniques can be incorporated into our framework to improve the performance of similarity checking with MDs.

Matching rules are studied in [Bertossi et al. 2011; Fan et al. 2011a; Hernandez and Stolfo 1998] (positive) and [Arasu et al. 2009; Whang et al. 2009] (negative). Data repairing was first studied in [Arenas et al. 2003; Fellegi and Holt 1976]. A variety of constraints have been used to specify data consistency in data repairing, *e.g.,* FDs [Wijsen 2005], FDs and INDs [Bohannon et al. 2005], and CFDs [Cong et al. 2007; Fan et al. 2008] (see [Fan 2008] for a survey). In this work, we employ CFDs, and extend MDs of [Fan et al. 2011a] with negative rules.

The consistency and implication problems have been studied for CFDs [Fan et al. 2008] and MDs [Fan et al. 2011a] separately. In this work we study these problems for MDs and CFDs put together. It is known that data repairing is NP-complete [Bohannon et al. 2005; Cong et al. 2007]. 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 [Bohannon et al. 2005; Cong et al. 2007].

Several repairing algorithms have been proposed [Bohannon et al. 2005; Cong et al. 2007; Fan et al. 2010; Fellegi and Holt 1976; Mayfield et al. 2010; Yakout et al. 2010]. Heuristic methods are developed in [Bohannon et al. 2005; Cong et al. 2007; Fellegi and Holt 1976], based on FDs and INDs [Bohannon et al. 2005], CFDs [Fan et al. 2008], and edit rules [Fellegi and Holt 1976]. The methods of [Bohannon et al. 2005; Cong et al. 2007] employ confidence placed by users to guide a repairing process. Statistical inference is studied in [Mayfield et al. 2010] to derive missing values. To ensure the accuracy of repairs generated, [Mayfield et al. 2010; Yakout et al. 2010] 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 [Fan et al. 2010], also based on master data. It differs from our work in the following. (i) While [Fan et al. 2010] 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 [Fan et al. 2010] 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 [Weis and Naumann 2005; Dong et al. 2005; Whang et al. 2009; Guo et al. 2010]. Among these, (1) [Guo et al. 2010] proposes to use uniqueness constraints to cluster objects from multiple data sources, and employs machine learning techniques to discover the true values of the objects; it differs from this work in the set of constraints used; and (2) [Weis and Naumann 2005; Dong et al. 2005; Whang et al. 2009] investigate record matching in the presence of errors in the data, and advocate the need for data repairing to match

records. The merge/fusion operations adopted there are more restrictive than updates (value modifications) suggested by cleaning rules of this work. Furthermore, when no matches are found, no merge or fusion can be conducted, whereas this work may still repair data with CFDs.

There has also been a host of work on ETL tools (see [Herzog et al. 2009] for a survey), which support data transformations, and can be employed to merge and fix data [Naumann et al. 2006], although they are typically not based on a constraint theory, and focus on syntactic errors in specific domains instead of semantic errors [Rahm and Do 2000]. These are complementary to data repairing and this work.

Information entropy measures the degree of uncertainty [Cover and Thomas 1991]: the smaller 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 [Srivastava and Venkatasubramanian 2010]. 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

We first review CFDs [Fan et al. 2008], which specify the consistency of data for data repairing. We then extend MDs [Fan et al. 2011a] 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 (see *e.g.,* [Fan et al. 2011b; Song and Chen 2009]).

2.1. Conditional Functional Dependencies

Following [Fan et al. 2008], we define a *conditional functional dependency* (CFD) φ on relation instances of schema R as a pair $R(X \to Y, t_p)$, where

(1) $X \to Y$ is a standard FD on instances of 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 left-hand side (LHS) and right-hand side (RHS) of φ, respectively.

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

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

CFD φ_2 differs from φ_1 only in its pattern tuple. Note that FDs are a special case of CFDs in which pattern tuples consist of only wildcards, *e.g.,* φ_3 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, $D \not\models \varphi_4$, since t_3 does not satisfy φ_4. Intuitively, CFD φ_4 says that no tuple t can have $t[\mathsf{FN}]$ = Bob (it has to be changed to Robert). In contrast, $D \models \varphi_3$, since 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 [Fan et al. 2011a; Hernandez and Stolfo 1998], we define matching dependencies (MDs) in terms of a set Υ of similarity predicates, *e.g.,* q-grams, Jaro distance or edit distance (see [Elmagarmid et al. 2007] 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 one or more unreliable data sources (see [Fan et al. 2011a] for a detailed discussion of their dynamic semantics). In contrast, we focus on matching tuples across a dirty data source D and a master relation D_m. To handle 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 $j \in [1, k]$, the attribute values $t[A_j]$ and $s[B_j]$ are similar, *i.e.,* $t[A_j] \approx_j s[B_j]$, then $t[E_i]$ is *changed* to $s[F_i]$, using values drawn from 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 updated with master tuples in D_m.

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

Negative MDs. Along the same lines as [Arasu et al. 2009; Whang et al. 2009], 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 as the same entity.

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

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

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. a master relation* 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. positive MD) ξ, we use LHS(ξ) and RHS(ξ) to denote the LHS and RHS of ξ, respectively. It is called *normalized* if |RHS(ξ)| = 1, *i.e.,* its right-hand side consists of a single attribute (resp. attribute pair).

As shown in [Fan et al. 2008; Fan et al. 2011a], every CFD ξ (resp. positive MD) can be expressed as an equivalent set S_ξ of normalized CFDs (resp. positive MDs), such that the cardinality of S_ξ is bounded by the size of RHS(ξ). For instance, CFDs φ_1, φ_2 and φ_4 of Example 1.1 are normalized. While φ_3 is not normalized, it can be converted to an equivalent set of CFDs of the form $([\mathsf{city}, \mathsf{phn}] \rightarrow A_i, t_{p_i})$, where A_i ranges over St, AC and post, and t_{p_i} consists of wildcards only; similarly for MD ψ.

Embedding negative MDs. We say that a set Γ of MDs is *equivalent* to another set Γ' of MDs, denoted by $\Gamma \equiv \Gamma'$, if for any instance D of R, $(D, D_m) \models \Gamma$ iff $(D, D_m) \models \Gamma'$.

The following example suggests that negative MDs can be converted to equivalent positive MDs. As a result, there is no need to treat them separately.

Example 2.5. Consider MD ψ in Example 1.1 and negative MD ψ^- in Example 2.4. We define ψ' by incorporating the premise (gd) of ψ^- into the premise of ψ:

ψ': tran[LN, city, St, post, gd] = card[LN, city, St, zip, gd] $\wedge$ tran[FN] $\approx$ card[FN] $\rightarrow$ tran[FN, phn] $\rightleftharpoons$ card[FN, tel].

Then no tuples with different genders can be identified as the same person, which is precisely what ψ^- aims 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 2.6: *Given a nonempty set Γ_m^+ of positive MDs and a set Γ_m^- of negative MDs, there exists a set Γ_m of positive MDs that is equivalent to $\Gamma_m^+ \cup \Gamma_m^-$ and can be computed in $O(|\Gamma_m^+||\Gamma_m^-|)$ time.* □

PROOF. We show this by presenting an algorithm that given as input Γ_m^+ and Γ_m^-, computes Γ_m that is equivalent to $\Gamma_m^+ \cup \Gamma_m^-$, in $O(|\Gamma_m^+||\Gamma_m^-|)$ time.

(1) We first present the algorithm. Assume *w.l.o.g.* that each MD in Γ_m^+ is normalized.
(a) We use a set Γ_m, initially empty.
(b) For each positive MD $\psi = \text{LHS}(\psi) \rightarrow (R[E] \rightleftharpoons R_m[F])$ in Γ_m^+, a new positive MD ψ' is generated as follows.
- First, let $L = \text{LHS}(\psi)$.
- Second, *for each* negative MD ψ^- in the form of $\bigwedge_{j\in[1,k]}(R[A_j] \neq R_m[B_j]) \rightarrow (R[E] \not\rightleftharpoons R_m[F])$ in Γ_m^-, let $L = \bigwedge_{j\in[1,k]}(R[A_j] = R_m[B_j]) \bigwedge L$.
- Finally, after all negative MDs in Γ_m^- are processed, let $\psi' = L \rightarrow (R[E] \rightleftharpoons R_m[F])$ and $\Gamma_m = (\Gamma_m \setminus \{\psi\}) \cup \{\psi'\}$.

(c) Return Γ_m after all positive MDs in Γ_m^+ are processed.

(2) We then show that the algorithm is correct, *i.e.,* (a) it is in $O(|\Gamma_m^+||\Gamma_m^-|)$ time and (b) $\Gamma_m \equiv \Gamma_m^+ \cup \Gamma_m^-$. For (a), observe that (i) each positive MD in Γ_m^+ is scanned once, and (ii) for each positive MD, all negative MDs are scanned once. This tells us that the algorithm above indeed runs in $O(|\Gamma_m^+||\Gamma_m^-|)$ time. For (b), one can easily verify that Γ_m is equivalent to $\Gamma_m^+ \cup \Gamma_m^-$ by induction on the number of negative MDs in Γ_m^-. □

In light of these, in the sequel we consider normalized CFDs and positive MDs only.

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 $\Theta = \Sigma \cup \Gamma$ of data quality rules, where Σ is a set of CFDs defined on R, and Γ is a set of MDs defined 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. Following [Arenas et al. 2003], we state the *data cleaning problem*, referred to as DCP, as follows. It takes D, D_m and Θ as input, and computes 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) D_r should be *consistent*, (b) no more tuples in D_r can be updated by *matching* rules, and (c) D_r is accurate and is close to the original data D. Following [Cong et al. 2007], 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) $t[A].\mathsf{cf}$ is the *confidence* placed by the user in the accuracy of the attribute $t[A]$ (see the cf rows in Fig. 1(b)).

The quality metric says that *the higher the confidence* of 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, *e.g.,* longer strings with 1-character difference are closer than shorter strings with 1-character difference. As shown in [Cong et al. 2007], confidence can be derived via provenance analysis. For instance, information that has been inspected and approved by domain experts has higher confidence than unreviewed data predicted by programs [Bolleman et al. 2010]. This is reinforced by recent work on determining the reliability of data sources (*e.g.,* [Dong et al. 2010]).

Cleaning rules. A variety of integrity constraints have been studied for data repairing (*e.g.,* [Bohannon et al. 2005; Cong et al. 2007; Fan et al. 2008; Wijsen 2005]). As observed by [Fan et al. 2010], while these constraints help us determine whether data is dirty, *i.e.,* whether errors are present, 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 [Klir and Folger 1988].

(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) $t[C].\mathsf{cf} := d$ for each $C \in E$, where d is the minimum $t[A_j].\mathsf{cf}$ for all $j \in [1, k]$ if $\approx_j$ is '='.

That is, the cleaning rule γ_ψ corrects $t[E]$ with the clean master value $s[F]$, and infers the new confidence of $t[E]$ following fuzzy logic [Klir and Folger 1988]. Intuitively, the confidence of an attribute value is about how confident a variable is correct (*i.e.,* in the concept of fuzzy set membership), instead of how probable a variable is correct (*i.e.,* in

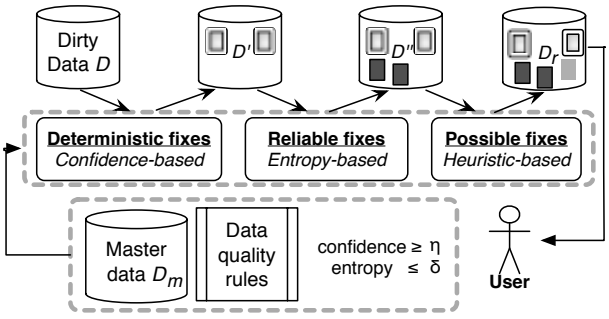


Fig. 2. Framework Overview

the concept of subjective probability). Hence, we update the confidence by taking the minimum rather than the product of the confidences in the premise of the MDs.

(2) Constant CFDs. Consider a CFD $\varphi_c = R(X \to A, t_{p_1})$, 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) $t[A].\mathsf{cf} = d$. Here d is the minimum $t[A'].\mathsf{cf}$ for all $A' \in X$ along the same lines as above. That is, the cleaning rule corrects $t[A]$ with the constant in the CFD.

(3) Variable CFDs. Consider a CFD $\varphi_v = (Y \to B, t_{p_2})$, 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 of $t_1[B'].\mathsf{cf}$ and $t_2[B'].\mathsf{cf}$ for all $B' \in Y$.

While cleaning rules derived from MDs are similar to editing rules of [Fan et al. 2010], rules derived from (constant or variables) CFDs are not studied there. We use confidence and infer new confidences based on fuzzy logic [Klir and Folger 1988]. There are various ways to enforce the dependency, which will be discussed shortly.

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 3.1. 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. 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 cleaning process. □

3.2. A Tri-level Data Cleaning Solution

Based on cleaning rules, we develop a data cleaning system, referred to as 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 [Bohannon et al. 2005; Cong et al. 2007; Fan et al. 2010; Fellegi and Holt 1976; Mayfield et al. 2010; Yakout et al. 2010], UniClean generates fixes by unifying matching and repairing processes, via cleaning rules. Furthermore, it stresses the accuracy by distinguishing these fixes with three levels of accuracy. Indeed, various fixes are found by executing three algorithms consecutively, as shown in Fig. 2 and illustrated below.

(1) Deterministic fixes based on confidences. It identifies erroneous attributes $t[A]$ to which there exists a fix with high confidence, referred to as a *deterministic fix*. It corrects those errors based on confidence: it uses a cleaning rule γ to update $t[A]$ *only if* all the attributes of t in the premise of γ have confidence above the threshold η, and moreover, $t[A]$ is changed to a (master data) value v on which the confidence is also above η. It is evident that such fixes are accurate up to η. Moreover, when the rule and the (master) data value v are asserted correct, the fix to $t[A]$ is typically unique (see Section 5 for details).

(2) Reliable fixes based on entropy. For the 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 [Hamming 1950] and compression [Ziv and Lempel 1978], 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 given 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 [Cong et al. 2007], by supporting cleaning rules derived from both CFDs and MDs. It can be verified that the heuristic method always finds a repair D_r of D such that $D_r \models \Sigma$, $(D_r, D_m) \models \Gamma$, while keeping all the deterministic fixes produced earlier *unchanged*.

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, followed by a brief discussion of possible fixes in Section 7.

Remark. There is no need to iterate the processes for the three types of fixes. Indeed, after the process for identifying reliable fixes, no fixes can be deterministic since the latter is based on the outcome of the former process, and reliable fixes may contain errors. Moreover, after the process to find heuristic fixes, all inconsistencies are fixed and hence, no inconsistencies may be detected by the rules anymore.

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, as will be seen in later sections.

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, a master relation D_m, and a set $\Theta = \Sigma \cup \Gamma$ of CFDs and MDs, as stated in Section 3.

4.1. Reasoning about Data Quality Rules

There are two classical problems associated with data quality rules.

Consistency. 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*. Rules may be manually designed by human experts or automatically discovered by data cleaning tools [Fan et al. 2011b; Song and Chen 2009]. In either case, it is common

that the rules may contain mistakes. This highlights the need for the consistency analysis, since it does not make sense to derive cleaning rules from Θ before Θ is assured consistent itself.

Implication. We say that Θ *implies* another CFD (resp. MD) ξ, denoted by $\Theta \models \xi$, 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, Θ and another CFD (or MD) ξ, whether $\Theta \models \xi$.

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

The consistency and implication 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 [Fan et al. 2011a]. In contrast, there exist CFDs that are inconsistent, and the consistency analysis of CFDs is NP-complete [Fan et al. 2008]. It is also known that the implication problem for MDs is in quadratic time [Fan et al. 2011a], whereas the implication problem for CFDs is coNP-complete [Fan et al. 2008]. Below we show that these problems for CFDs and MDs put together have the same complexity as their counterparts for CFDs. That is, adding MDs to CFDs does not make our lives harder.

THEOREM 4.1. *For CFDs and MDs put together, the consistency problem remains NP-complete.*

PROOF. The lower bound follows from the intractability of its CFD counterpart, which is NP-complete [Fan et al. 2008]. Indeed, the consistency problem for CFDs is a special case of the consistency problem for CFDs and MDs put together, when $\Gamma = \emptyset$.

To show that the problem is in NP, consider a master relation D_m of schema R_m, a set Σ of CFDs on schema R, and a set Γ of MDs on (R, R_m). The upper bound is verified by establishing a *small model property*: if $\Sigma \cup \Gamma$ is consistent, then there exists a single-tuple instance $D = \{t\}$ such that $D \models \Sigma$ and $(D, D_m) \models \Gamma$. Indeed, if there exists a nonempty instance D of R such that (D, D_m) satisfies $\Sigma \cup \Gamma$, then for any tuple $t \in D$, $D_t = \{t\}$ is an instance of R such that (D_t, D_m) satisfies $\Sigma \cup \Gamma$. Hence it suffices to consider single-tuple instances $D = \{t\}$ for deciding whether $\Sigma \cup \Gamma$ is consistent.

Assume *w.l.o.g.* that the set $\mathsf{attr}(R)$ of attributes in R is $\{A_1, \ldots, A_n\}$. Moreover, for each $i \in [1, n]$, let the active domain $\mathsf{adom}(A_i)$ of A_i consist of all constants appearing in Σ and D_m, plus at most an extra distinct value drawn from the domain $\mathsf{dom}(A_i)$ of A_i, if such a value exists (*e.g.,* when $\mathsf{dom}(A_i)$ is a finite domain). Then it is easy to verify that $\Sigma \cup \Gamma$ is consistent iff there exists a mapping ρ from $t[A_i]$ to $\mathsf{adom}(A_i)$ for each $i \in [1, n]$ such that $D' = \{(\rho(t[A_1]), \ldots, \rho(t[A_n]))\}$, $D' \models \Sigma$ and $(D', D_m) \models \Gamma$.

Along the same lines as the proof of CFDs [Fan et al. 2008], we give an NP algorithm for checking the consistency of $\Sigma \cup \Gamma$ as follows: (a) Guess a single tuple t of R such that $t[A_i] \in \mathsf{adom}(A_i)$. (b) Check whether $D = \{t\}$ and D_m satisfy $\Sigma \cup \Gamma$. Obviously step (b) can be done in PTIME in the size $|\Sigma \cup \Gamma|$ of $\Sigma \cup \Gamma$. Thus this algorithm is in NP.

Hence the consistency problem is in NP when CFDs and MDs are put together. □

THEOREM 4.2. *For CFDs and MDs put together, the problem for deciding whether $\Sigma \cup \Gamma$ implies ψ remains coNP-complete, when ψ is either a CFD or an MD.*

PROOF. We first show that the problem is in coNP, and then we show that the problem is coNP-hard.

(I) We first consider CFD $\psi = (X \rightarrow A, t_p)$. Similar to the upper bound proof of Theorem 4.1, the upper bound is verified by establishing a small model property: if $\Sigma \cup \Gamma \not\models \psi$, then there exists a relation D of R such that D consists of two tuples t, s such that $t[X] = s[X] \asymp t_p[X]$, $D \models \Sigma$ and $(D, D_m) \models \Gamma$, but $(D, D_m) \not\models \psi$. Here for each attribute $A \in X \cup Y$, $s[A] \in \mathsf{adom}(A)$ and $t[A] \in \mathsf{adom}(A)$. Then an NP algorithm similar

to the one given in the proof of Theorem 4.1 suffices to check whether there exists a relation D' consisting of two tuples such that $D' \models \Sigma$, $(D', D_m) \models \Gamma$ and $D' \not\models \psi$.

The proof is similar when ψ is an MD. If $\Sigma \cup \Gamma \not\models \psi$, then there exists a relation D of R such that D consists of a single tuple t, where $D \models \Sigma$ and $(D, D_m) \models \Gamma$, but $(D, D_m) \not\models \psi$. An NP algorithm can be given for this case. Thus the problem is in coNP.

(II) When ψ is a CFD, the lower bound follows from the intractability of its CFD counterpart, which is known to be coNP-complete [Fan et al. 2008]. Indeed, the implication problem for CFDs is a special case of the implication problem for CFDs and MDs put together, when the set Γ of MDs is empty.

We next show that the problem is coNP-hard when ψ is an MD. It suffices to show that the complement of the problem is NP-hard by reduction from the 3SAT problem, which is NP-complete (cf. [Garey and Johnson 1979]). An instance ϕ of 3SAT is of the form $C_1 \wedge \cdots \wedge C_n$, where all the variables in ϕ are $x_1, \ldots, x_m$, each clause C_j ($j \in [1, n]$) is of the form $y_{j_1} \vee y_{j_2} \vee y_{j_3}$, and moreover, y_{j_i} is either $x_{p_{ji}}$ or $\overline{x_{p_{ji}}}$ for $p_{ji} \in [1, m]$, for $i \in [1, 3]$. Here we use $x_{p_{ji}}$ to denote the occurrence of a variable in the literal i of clause C_j. The 3SAT problem is to determine whether ϕ is satisfiable.

Given an instance ϕ of the 3SAT problem, we construct two relational schemas R and R_m, a set of Σ of CFDs defined on R, a set of Γ of MDs on (R, R_m), a master relation D_m of R_m, and another MD ψ such that $\Sigma \cup \Gamma \models \psi$ iff the 3SAT instance ϕ is *not* satisfiable.

(1) We first build the instance of the implication problem.

- The two schemas are $R(X_1, \ldots, X_m, Z)$ and $R_m(Z)$, respectively, in which all the attributes have a *Boolean* domain $\{T, F\}$. Intuitively, for each R tuple t, $t[X_1, \ldots, X_m]$ and $t[Z]$ encode a truth assignment for the variables $x_1, \ldots, x_m$ of the 3SAT instance ϕ and the truth value of ϕ under the assignment, respectively.
- The set Γ is empty.
- The set Σ consists of n CFDs, defined as follows. For each clause $C_j = y_{j_1} \vee y_{j_2} \vee y_{j_3}$ of ϕ ($j \in [1, n]$), we define a CFD $\psi_j = (X_{p_{j_1}} X_{p_{j_2}} X_{p_{j_3}} \to Z, t_{p,j})$, where $t_{p,j}[X_{p_{j_1}} X_{p_{j_2}} X_{p_{j_3}}]$ is the unique truth assignment that makes C_j false, and $t_{p,j}[Z] = F$.
- The master relation D_m consists of a single tuple s such that $s[Z] = (F)$.
- The MD ψ is $\emptyset \to (R[Z] \rightleftharpoons R_m[Z])$.

(2) We next show that $\Sigma \cup \Gamma \models \psi$ iff the 3SAT instance ϕ is *not* satisfiable.

Assume first that $\Sigma \cup \Gamma \models \psi$. We show that ϕ is *not* satisfiable. We assume that ϕ is satisfiable by contradiction. Then there exists a truth assignment ξ of ϕ that makes ϕ true. Let D be an instance of R that consists of a single tuple t such that $t[X_1, \ldots, X_m] = \xi[x_1, \ldots, x_m]$ and $t[Z] = T$. It is easy to verify that $D \models \Sigma$ and $(D, D_m) \models \Gamma$, but $(D, D_m) \not\models \psi$. This contradicts the assumption that $\Sigma \cup \Gamma \models \psi$. Hence ϕ is *not* satisfiable.

Conversely, we assume that ϕ is *not* satisfiable. We show that $\Sigma \cup \Gamma \models \psi$. For any instance D of R, if $D \models \Sigma$ $(D, D_m) \models \Gamma$, then for any tuple $t \in D$, $t[X_1, \ldots, X_m]$ is a truth assignment that makes ϕ false. Hence there must exist a CFD $\psi_j = (X_{p_{j_1}} X_{p_{j_2}} X_{p_{j_3}} \to Z, t_{p,j})$ (*i.e.,* there exists $C_j = y_{j_1} \vee y_{j_2} \vee y_{j_3}$, for some $j \in [1, n]$) such that $t_{p,j}[X_{p_{j_1}} X_{p_{j_2}} X_{p_{j_3}}] = t[X_{p_{j_1}} X_{p_{j_2}} X_{p_{j_3}}]$ and $t[Z] = t_{p,j}[Z]$ = false. By the definition of ψ, $t[Z] = F$ and $(D, D_m) \models \psi$. Hence we have that $\Sigma \cup \Gamma \models \psi$.

Taken together, (I) and (II) show that the implication problem is coNP-complete for CFDs and MDs put together, when the data quality rule ψ is either a CFD or an MD. □

In the sequel 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 a repair D_r with $\mathsf{cost}(D_r, D)$ below a predefine bound. Worse still, it is infeasible in practice to find a 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 4.3. *(1) The data cleaning problem (DCP) is NP-complete. (2) Unless P = NP, for any constant ϵ, there exists no PTIME ϵ-approximation algorithm for DCP.*

PROOF. (1) This can be easily verified along the same lines as for FDs [Bohannon et al. 2005] or CFDs [Fan et al. 2008]. The upper bound is verified by giving an NP algorithm, and the lower bound is obvious since the problem is already NP-hard for CFDs only [Cong et al. 2007].

(2) This is verified by reduction from 3SAT, using gap techniques [Wegener and Pruim 2005]. Given any constant $\epsilon > 1$ and any 3SAT instance ϕ as described in the proof of Theorem 4.2, we first construct an instance of DCP that consists of two schemas R and R_m, a set of Σ of CFDs defined on R, a set of Γ of MDs on (R, R_m), an instance D_m of R_m, and an instance D of R such that $D \not\models \Sigma$ or $(D, D_m) \not\models \Gamma$. Assume *w.l.o.g.* that D_o is the repair of D with optimal cost such that $D_o \models \Sigma$ and $(D_o, D_m) \models \Gamma$. We then show that there exists an algorithm for DCP that finds a repair D_r of D such that $\mathsf{cost}(D_r, D) \leq \epsilon \cdot \mathsf{cost}(D_o, D)$ iff there exists a PTIME algorithm to determine whether 3SAT instances ϕ are satisfiable. From these it follows that unless P = NP, for any constant ϵ, there exists no PTIME ϵ-approximation algorithm for DCP. Note that here ϵ must be larger than 1 as DCP is a minimization problem.

(a) We first construct the DCP instance.

- We define schema $R(X_1, \ldots, X_m, C_{1,1}, \ldots, C_{1,r}, \ldots, C_{n,1}, \ldots, C_{n,r})$, where $r = \lceil \epsilon \cdot m \rceil + 1$, *i.e.,* the second smallest positive integer no less than $\epsilon \cdot m$, and all the attributes in R have a Boolean domain $\{T, F\}$. Intuitively, for each R tuple t, $t[X_1, \ldots, X_m]$ and $t[C_{j,1}] = \ldots = t[C_{j,r}]$ ($j \in [1, n]$) encode a truth assignment for the variables $x_1, \ldots, x_m$ of the 3SAT instance ϕ and the truth value of clause C_j under the assignment, respectively.
- The set Σ consists of n CFDs given as follows. For each clause C_j ($j \in [1, n]$) of the form $y_{j_1} \vee y_{j_2} \vee y_{j_3}$ in ϕ, we define a traditional FD $\psi_j = X_{p_{j_1}} X_{p_{j_2}} X_{p_{j_3}} \rightarrow C_{j,1} \ldots C_{j,r}$. Note that FDs are a special case of CFDs.
- The instance D consists of $n + 1$ tuples. For each clause C_j ($j \in [1, n]$) of the form $y_{j_1} \vee y_{j_2} \vee y_{j_3}$ in ϕ, let ξ_j be the unique truth assignment that makes clause C_j false, and $\xi_j(x) = T$ for all remaining variables x not in clause C_j. We then define a tuple t_j of R in terms of ξ_j such that (a) for each $i \in [1, m]$, $t_j[X_i] = \xi_j(x_i)$, (b) $t_j[C_{j,1}, \ldots, C_{j,r}] = (F, \ldots, F)$, and (c) for each remaining clause C_i of ϕ ($i \neq j \wedge i \in [1, n]$), $t_j[C_{i,1}] = \ldots = t_j[C_{i,r}]$ is the corresponding truth value under the assignment ξ_j.
  We further add to D an extra tuple t such that (a) for each $j \in [1, n]$, $t[C_{j,1}, \ldots, C_{j,r}] = (T, \ldots, T)$, and (b) for each $i \in [1, m]$, $t[X_i]$ is randomly assigned to T or F.
- The schema of R_m is arbitrary since it is irrelevant, and the set Γ of MDs is empty.

Assume that all attributes have a fixed default confidence cf. Observe that to find a repair D_r of D with minimum cost, it is necessary to change tuple t, rather than the other tuples t_j ($j \in [1, n]$) in D.

(b) We then show that there exists an algorithm with approximation ratio ϵ for the DCP instance iff there exists a PTIME algorithm for the 3SAT instance ϕ.

Assume first that the 3SAT instance ϕ is satisfiable. If there exists a polynomial time approximation algorithm with approximation ration ϵ, then we can find a tuple t' that disagrees with t in less than $\lceil \epsilon \cdot \mathsf{opt} \rceil$ fields, where opt is the optimum cost, and it is

simply the number of modified fields of t. Indeed, since ϕ is satisfiable, there exists a truth assignment ξ of variables $x_1, \ldots, x_m$ that makes ϕ true. From this, it is easy to see that opt $\leq m$. This implies that we can find a tuple t' in polynomial time by updating $t[X_1, \ldots, X_m]$ to $\xi[x_1, \ldots, x_m]$ such that (i) the tuple t' differs from tuple t in less than $\lceil \epsilon \cdot m \rceil$ fields, and moreover, (ii) $D \setminus \{t\} \cup \{t'\} \models \Sigma$.

Conversely, assume that the 3SAT instance ϕ is *not* satisfiable. Then for all truth assignments of variables $x_1, \ldots, x_m$ in the instance ϕ, there exists a clause C_j ($1 \leq j \leq n$) such that $\xi(C_j)$ is false. Thus, for each tuple t' such that $D \setminus \{t\} \cup \{t'\} \models \Sigma$, tuple t' must disagree with tuple t in at least $\lceil \epsilon \cdot m \rceil$ + 1 fields.

Putting these together, we have a PTIME algorithm for determining whether the 3SAT instance ϕ is satisfiable. If we can find a tuple t' such that (a) tuple t' differs from tuple t in less than $\lceil \epsilon \cdot m \rceil$ fields, and (b) $D \setminus \{t\} \cup \{t'\} \models \Sigma$, then the instance ϕ is satisfiable. If we can find a tuple t' such that (a) tuple t' differs from tuple t in at least $\lceil \epsilon \cdot m \rceil$ + 1 fields, and (b) $D \setminus \{t\} \cup \{t'\} \models \Sigma$, then the instance ϕ is *not* satisfiable. This contradicts the fact that the 3SAT problem is NP-complete. Therefore, unless P = NP, for any constant ϵ, there exists no PTIME ϵ-approximation algorithm for DCP. □

The proof of Theorem 4.3 reveals the following.

Corollary 4.4: *Unless P = NP, for any constant ϵ, there exists no PTIME ϵ-approximation algorithm for DCP, even for CFDs only, in the absence of MDs.* □

Proposition 4.5: *Unless P = NP, for any constant ϵ, there exists no PTIME ϵ-approximation algorithm for DCP, even for MDs only, in the absence of CFDs.* □

PROOF. This can also be verified by reduction from 3SAT, using gap techniques [Wegener and Pruim 2005]. The proof is similar to the one of Theorem 4.3. The only difference is the DCP instance, which is constructed as follows.

- The schema of R is the same as the one in the proof of Theorem 4.3.
- The instance D of R consists of a single tuple, the same as the R tuple t in the proof of Theorem 4.3.
- The schema R_m is the same as R.
- The master data D_m consists of n tuples, the same as the R tuples $\{t_1, \ldots, t_n\}$ in the proof of Theorem 4.3.
- The set Γ consists of n MDs. For each FD $X_{p_{j_1}} X_{p_{j_2}} X_{p_{j_3}} \rightarrow C_{j,1} \ldots C_{j,r}$ in the set Σ of CFDs in the proof of Theorem 4.3, we create an MD $R[X_{p_{j_1}} X_{p_{j_2}} X_{p_{j_3}}] = R_m[X_{p_{j_1}} X_{p_{j_2}} X_{p_{j_3}}] \rightarrow R[C_{j,1} \ldots C_{j,r}] \rightleftharpoons R_m[C_{j,1} \ldots C_{j,r}]$ in Γ .

Following a similar argument to the one in the proof of Theorem 4.3, we have that unless P = NP, for any constant ϵ, there exists no PTIME ϵ-approximation algorithm for DCP in the absence of CFDs. □

It is known that data repairing alone is already NP-complete for FDs [Bohannon et al. 2005]. Theorem 4.3 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 arising from rule-based data cleaning methods such as the one proposed in Section 3.

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.

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 *nondeterministic*: multiple rules can be applied at the same time. We want to know that whether the output of the process is independent of the order of the rules applied.

Worse still, it is known that even for repairing only, a rule-based method may lead to an *infinite* process [Cong et al. 2007].

Example 4.6. Consider CFD $\varphi_1 = \mathsf{tran}([\mathsf{AC}] \rightarrow [\mathsf{city}], t_{p_1} = (131 \parallel \text{Edi}))$ given in Example 2.1, and 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 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 4.7. *The termination problem is PSPACE-complete for rule-based data cleaning based on CFDs and MDs.*

PROOF. We first verify the lower bound of the problem by reduction from the halting problem for linear bound automata, which is PSPACE-complete [Aiken et al. 1993]. We then show the upper bound by providing an algorithm, which uses polynomial space in the size of input.

(I) We first show that the problem is PSPACE-hard by reduction from the halting problem for linear bound automata. A linear bounded automaton is a 6-tuple $\langle Q, \Upsilon, \Lambda, q_0, F, \delta \rangle$, where (a) Q is a finite set of states, (b) Υ is a finite tape alphabet, (c) $\mathsf{blank} \in \Upsilon$ is the blank symbol, (d) $\Lambda \subseteq (\Upsilon \setminus \{\mathsf{blank}\})$ is a finite input alphabet, (e) $q_0 \in Q$ is the start state, (f) $F \subseteq Q$ is the set of halting states, and (g) $\delta : Q \times \Upsilon \rightarrow Q \times \Lambda \times \{\mathsf{Left}, \mathsf{Right}\}$ is the next move function. Given a linear bounded automaton M with input alphabet Λ and a string $x \in \Lambda^*$, the halting problem asks whether M halts on input x.

Given an automaton M and input string x of length n, we first construct an instance for the termination problem: two schemas R_m (master relation) and R, a set Σ of CFDs on R, a set Γ of MDs on (R, R_m), an instance D_m of R_m, and an instance D of R. We then show that M accepts x if and only if the repairing process terminates for the constructed instance of the termination problem.

(1) The schema is $R(A, I, A_b, A_{nxt}, I_b, I_{nxt}, C_b, C_{nxt}, A_1, \ldots, A_n)$, where $n = |x|$, *i.e.,* the length of the input string x. All R attributes have finite domains: (a) attributes A, A_b and A_{nxt} have finite domain Q; (b) attributes I, I_b and I_{nxt} have finite domain $\{1, \ldots, n\}$; and (c) all the remaining attributes have finite domain Υ.

Intuitively, for an R tuple t, (a) $t[A_1, \ldots, A_n]$ denotes the current tape content of M, initially the input string x, (b) $t[A]$ is the current state of M, (c) $t[I]$ is the current position of the tape head of M, and, moreover, (d) $t[A_b]$, $t[A_{nxt}]$, $t[I_b]$, $t[I_{nxt}]$, $t[C_b]$, $t[C_{nxt}]$ are used to help encode the transition function δ of M, as will be seen shortly.

(2) The relation $D = \{t\}$, where $t = (q_0, 1, q_0, q_0, 1, 1, \mathsf{blank}, \mathsf{blank}, x[1], \ldots, x[n])$, where $x[i]$ is the i-th symbol in the input string x.

(3) The set Σ consists of $6(n-1)$ CFDs.

- For each $q \in Q \setminus F$ and $s \in \Upsilon$ such that $\delta(q \times s) = q' \times s' \times \mathsf{Left}$, Σ includes a set of $3(n-1)$ CFDs, given as follows. For each $i \in [2, n]$,
  $\varphi_{(L,i,1)} = R(AA_iI \rightarrow A_bC_bI_b, (q, s, i \parallel q, s, i))$,
  $\varphi_{(L,i,2)} = R(AA_iIA_bC_bI_b \rightarrow A_{nxt}C_{nxt}I_{nxt}, (q, s, i, q, s, i \parallel q', s', i-1))$, and
  $\varphi_{(L,i,3)} = R(A_bC_bI_bA_{nxt}C_{nxt}I_{nxt} \rightarrow AA_iI, (q, s, i, q', s', i-1 \parallel q', s', i-1))$.
  Intuitively, these CFDs together encode the transition $\delta(q \times s) = q' \times s' \times \mathsf{Left}$, by (a) updating the values of attributes A and A_i to q and s, respectively, and (b) decreasing the value of I by 1.
- For each $q \in Q \setminus F$ and $s \in \Upsilon$ such that $\delta(q \times s) = q' \times s' \times \mathsf{Right}$, Σ includes set of $3(n-1)$ CFDs. More specifically, for each $i \in [1, n-1]$,
  $\varphi_{(R,i,1)} = R(AA_iI \rightarrow A_bC_bI_b, (q, s, i \parallel q, s, i))$,

$\varphi_{(R,i,2)}$ = $R(AA_iIA_bC_bI_b \to A_{nxt}C_{nxt}I_{nxt}, (q, s, i, q, s, i \parallel q', s', i+1))$, and
$\varphi_{(R,i,3)}$ = $R(A_bC_bI_bA_{nxt}C_{nxt}I_{nxt} \to AA_iI, (q, s, i, q', s', i+1 \parallel q', s', i+1))$.
Similarly, these CFDs together encode the transition $\delta(q \times s)$ = $q' \times s' \times$Right, by (a) updating the values of attributes A and A_i to q and s, respectively, and (b) increasing the value of I by 1.

(4) The set Γ of MDs is empty. In addition, the master schema R_m and the master relation D_m are simply omitted as they are not needed in this reduction.

It is easy to verify that these CFDs indeed simulate the computation of a linear bounded automaton M. Hence, M halts on x iff the rule-based data cleaning process terminates on the constructed instance.

(II) We next show the upper bound of the terminating problem. Consider the data cleaning system UniClean discussed in Section 3.2, which could be treated as a PSPACE algorithm for the terminating problem. Observe that (a) the rule-based data cleaning process is non-deterministic, and (b) it uses space linear in the size of the input of the problem, *i.e.,* the size of the dirty relation D, master relation D_m, repair D_r and the cleaning rules Θ. By Savitch's theorem [Savitch 1970], there is a deterministic quadratic-space algorithm for the rule-based data cleaning process, and therefore, the termination problem is in PSPACE. □

THEOREM 4.8. *The determinism problem is PSPACE-complete for rule-based data cleaning based on CFDs and MDs.*

PROOF. We first show the upper bound by providing an algorithm, which uses polynomial space in the size of input. We then verify the lower bound of the problem by reduction from the halting problem for linear bound automata [Aiken et al. 1993].

(I) We show that the problem is in PSPACE, by presenting an NPSPACE algorithm that checks whether there are two terminating processes that yield distinct fixpoints. Given a master relation D_m of schema R_m, a (possibly) dirty relation D of schema R, a set Σ of CFDs on R, and a set Γ of MDs on (R, R_m), the algorithm works as follows:
(1) Guess two R instances D_1 and D_2 that both consist of only constants appearing in D, Σ and D_m, and both have the same number of tuples as D.
(2) Check whether $D_1 \neq D_2$, $D_1 \models \Sigma$, $(D_1, D_m) \models \Gamma$, $D_2 \models \Sigma$ and $(D_2, D_m) \models \Gamma$. If so, continue. Otherwise it rejects the guess and goes back to step (1).
(3) Check whether both D_1 and D_2 can be generated from D, by guessing and applying rules derived from Σ and Γ. That is, if the data cleaning process reaches a fixpoint for D, then it compares whether D_1 or D_2 is the same as the fixpoint. If so, it returns 'false' since distinct D_1 and D_2 can be generated from D. Otherwise it rejects the guess and goes back to step (1).

One can readily verify the correctness of the algorithm. That is, if it returns 'false', then the problem is not deterministic, and if the problem is not deterministic, then the algorithm returns 'false'. Note that there are finite number of R instances D as the rule-based data cleaning only uses constants appearing in D, Σ and D_m. Hence all the three steps above can be done in PSPACE, and the algorithm runs in PSPACE.

(II) We next show that the problem is PSPACE-hard by reduction from the halting problem for linear bound automata. The reduction is similar to its counterpart given in the proof of Theorem 4.7 for the termination problem, except that here we add the following CFDs, which assure that when the rule-based data cleaning process terminates, it always generates the same result. For each halting state $s \in F$, we include CFDs $\varphi_{(s,1)}$ = $R(A \to A_b, (s \parallel \natural))$ and $\varphi_{(s,2)}$ = $R(A_b \to \mathsf{attr}(R), (\natural \parallel \natural, \ldots, \natural))$. Here $\natural$ is a fresh distinct symbol. That is, all terminating processes end up with the same repair $(\natural, \ldots, \natural)$. Then

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 ($Y \to B$, t_p)

Fig. 3. Summary of notations

along the same lines as the proof for the termination problem given above, it can be verified that the determinism problem is PSPACE-hard. □

Moreover, along the same lines as Corollary 4.4 and Proposition 4.5, by encoding CFDs with MDs, one can easily verify the following.

Corollary 4.9: *The termination and determinism problems are both PSPACE-complete for rule-based data cleaning based on either CFDs or MDs only.* □

Remark. All the complexity results given above remain intact even in the absence of master data and matching dependencies (MDs). In particular, for CFDs alone, the data cleaning problem DCP is already NP-complete and approximation hard, and the termination problem and determinism problem are already PSPACE-complete. In other words, incorporating MDs and master data does not complicate data cleaning process.

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 Fig. 3 we also summarize some notations to be used in this Section and Section 6, for the ease of reference.

5.1. Deterministic Fixes

We define deterministic fixes *w.r.t.* a *confidence threshold* η, which may be determined by domain experts. When η is high enough, *e.g.,* if it is close to 1, an attribute $t[A]$ is considered 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 the 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 $\psi = \bigwedge_{j \in [1,k]} (R[A_j] \approx_j R_m[B_j]) \to (R[E] \rightleftharpoons R_m[F])$. 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 $\varphi_c = R(X \to A, t_{p_1})$. 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 $\varphi_v = (Y \to B, t_p)$. 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 [Abiteboul et al. 1995]. 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 tuple 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.

When master data, data cleaning rules and confidence levels placed on the data are all asserted correct, one can verify that the fix to $t[E]$ via an MD (resp. $t[A]$ via a constant CFD and $t_1[B]$ via a variable CFD) is unique, by the definition of deterministic fixes and along the same lines as [Fan et al. 2010]. In the sequel we assume the correctness of master data, data cleaning rules and confidence levels when studying deterministic fixes, to simplify the discussions.

Observe that when 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 discussions in Section 3.1). As a result, $t[A]$ is also asserted correct (up to confidence η), 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 this section we show that deterministic fixes can be found in PTIME, stated below.

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 prove Theorem 5.1 by providing an algorithm for deterministic fixes that runs in $O(|D||D_m|\mathsf{size}(\Theta))$ time. We now present the algorithm, followed by indexing structures and procedures that it employs.

Algorithm. The algorithm, referred to as cRepair, is shown in Fig. 4. It takes as input a set Σ of CFDs, a set Γ of MDs, master data D_m, a dirty relation D, and a confidence threshold η. It returns a partially cleaned repair D' with deterministic fixes marked.

Algorithm cRepair first initializes variables and auxiliary 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 rules derived from variable CFDs, constant CFDs, or MDs, respectively. It checks each tuple at most once *w.r.t.* each rule, makes more attributes asserted at each step, and uses these attributes to identify more deterministic fixes recursively. 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 marked (line 16).

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

Hash tables. We maintain a hash table for each variable CFD $\varphi = R(Y \to B, t_p)$, denoted as H_φ. Given an $\bar{y} \in \rho_{Y \asymp t_p[Y]}(D)$ 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 users is correct.

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: $\mathsf{vCFDInfer}(t,\ \xi,\ \eta)$;
13. (2) constant CFD: $\mathsf{cCFDInfer}(t,\ \xi,\ \eta)$;
14. (3) MD: $\mathsf{MDInfer}(t,\ \xi,\ \eta,\ , D_m)$;
15. **until** $\mathsf{Q}[t']$ is empty for any $t' \in D'$;
16. **return** D'.

Fig. 4. Algorithm cRepair

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 $\mathrm{LHS}(\xi)$. 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 $\mathsf{H}_\varphi(t[\mathrm{LHS}(\varphi)]).\mathsf{val}{=}\mathsf{nil}$, *i.e.,* no B attribute in $\Delta(t[\mathrm{LHS}(\varphi)])$ has a high enough confidence.

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

Procedures. We now present the procedures used in cRepair as shown in Fig. 5. Since those indexing structures are used in all these procedures, we simply denote the queue, hash set and counter as $\mathsf{Q}[t]$, $\mathsf{P}[t]$ and $\mathsf{count}[t, \xi]$, respectively, without explicitly specifying them as part of the input.

Procedure update. Given a new deterministic fix for $t[A]$, it propagates the change, in order to find the other deterministic fixes with $t[A]$. (a) For each rule ξ, if $A \in \mathrm{LHS}(\xi)$, $\mathsf{count}[t, \xi]$ is increased by 1 as one more attribute becomes asserted (lines 1-2). (b) If all attributes in $\mathrm{LHS}(\xi)$ are asserted, ξ is inserted into the queue $\mathsf{Q}[t]$ (line 3). (c) For a variable CFD $\xi' \in \mathsf{P}[t]$, if $\mathrm{RHS}(\xi')$ is A and $\mathsf{H}_{\xi'}(t[\mathrm{LHS}(\xi')]).\mathsf{val} = \mathsf{nil}$, the newly asserted $t[A]$ makes it possible for tuples in $\mathsf{H}_{\xi'}(t[\mathrm{LHS}(\xi')]).\mathsf{list}$ to have a deterministic fix. Thus ξ' is removed from $\mathsf{P}[t]$ and added to $\mathsf{Q}[t]$ (lines 4–5).

Procedure vCFDInfer. For a tuple t, a variable CFD ξ and a confidence threshold η, it finds a deterministic fix for t by applying ξ. If the tuple t and the pattern tuple $t_{(p,\xi)}$ match on their $\mathrm{LHS}(\xi)$ attributes, it does the following.

(a) If $t[\mathrm{RHS}(\xi)].\mathsf{cf} \geq \eta$ and no B-attribute values in $\mathsf{H}_\xi(t[\mathrm{LHS}(\xi)]).\mathsf{list}$ are asserted (line 1), it takes $t[\mathrm{RHS}(\xi)]$ as the B value in the set (line 2), and propagates the change via procedure update (lines 3–5).

(b) If $t[\mathrm{RHS}(\xi)] < \eta$ and there is an asserted B-attribute value val in $\mathsf{H}_\xi(t[\mathrm{LHS}(\xi)]).\mathsf{list}$

Procedure update(t, A)
Input: A new deterministic fix $t[A]$.
Output: Updated indexing structures.
1. **for** each $\xi \in \Sigma \cup \Gamma$ such that $A \in \text{LHS}(\xi)$ **do**
2. count$[t, \xi]$:= count$[t, \xi]$ + 1;
3. **if** count$[t, \xi]$ = $|\text{LHS}(\xi)|$ **then** Q$[t]$.push(ξ);
4. **for** each variable CFD $\xi' \in \Sigma \cap \text{P}[t]$ with $\text{RHS}(\xi') = A$ **do**
5. P$[t]$.remove(ξ');
6. **if** H$[\xi']$.get($t[\text{LHS}(\xi')]$).val $=$ nil **then** Q$[t]$.push(ξ');

Procedure vCFDInfer (t, ξ, η)
Input: Tuple t, variable CFD ξ, and confidence threshold η.
Output: Deterministic fixes and updated indexing structures.
1. **if** $t[\text{LHS}(\xi)] \asymp t_{(p,\xi)}[\text{LHS}(\xi)]$ **and** $t[\text{RHS}(\xi)]$.cf $\geq \eta$
and H$[\xi]$.get($t[\text{LHS}(\xi)]$).val $=$ nil **then**
2. H$[\xi]$.get($t[\text{LHS}(\xi)]$).val := $t[\text{RHS}(\xi)]$;
3. **for** each $t' \in$ H$[\xi]$.get($t[\text{LHS}(\xi)]$).list **do**
4. $t'[\text{RHS}(\xi)]$:= $t[\text{RHS}(\xi)]$; $t'[\text{RHS}(\xi)]$.cf := η;
5. update($t', \text{RHS}(\xi)$);
6. **else if** $t[\text{LHS}(\xi)] \asymp t_{(p,\xi)}[\text{LHS}(\xi)]$ **and** $t[\text{RHS}(\xi)]$.cf $< \eta$ **then**
7. **if** H$[\xi]$.get(t).val $\neq$ nil **then**
8. $t[\text{RHS}(\xi)]$:= H$[\xi]$.get($t[\text{LHS}(\xi)]$).val;
9. update($t, \text{RHS}(\xi)$); $t'[\text{RHS}(\xi)]$.cf := η;
10. **else** H$[\xi]$.get($t[\text{LHS}(\xi)]$).list.add(t); P$[t]$.add(ξ);

Procedure cCFDInfer (t, ξ, η)
Input: Tuple t, constant CFD ξ, and confidence threshold η.
Output: Deterministic fixes and updated indexing structures.
1. **if** $t[\text{LHS}(\xi)] \asymp t_{(p,\xi)}[\text{LHS}(\xi)]$ **then**
2. $t[\text{RHS}(\xi)] := t_{(p,\xi)}[\text{RHS}(\xi)]$; update($t, \text{RHS}(\xi)$);

Procedure MDInfer (t, ξ, η, D_m)
Input: Tuple t, constant CFD ξ, confidence threshold η and master data D_m.
Output: Deterministic fixes and updated indexing structures.
1. **if** $\exists t_m \in D_m$ ($t[\text{LHS}(\xi)] \asymp_\xi t_m[\text{LHS}(\xi)]$) **then**
2. $t[\text{RHS}(\xi)] := t_m[\text{RHS}(\xi)]$; update($t, \text{RHS}(\xi)$);

Fig. 5. Procedures of cRepair

(lines 6-7), it makes a deterministic fix by $t[\text{RHS}(\xi)] := \text{val}$ (line 8), and propagates the change via procedure update (line 9).

(c) If $t[\text{RHS}(\xi)] < \eta$ and there are no asserted B-attribute values in $\text{H}_\xi(t[\text{LHS}(\xi)])$.list, then there are no deterministic fixes that can be made at this moment. Hence, tuple t is simply added to $\text{H}_\xi(t[\text{LHS}(\xi)])$.list and P$[t]$, for later checking (line 10).

Procedures cCFDInfer *and* MDInfer. The first procedure takes as input a tuple t, a constant CFD ξ and a confidence 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 in Section 3.1. The changes made are propagated by invoking procedure update$(t, \text{RHS}(\xi))$.

Example 5.2. Consider master data D_m and relation D of Fig. 1, and cleaning rules Θ consisting of ξ_1, ξ_2 and ξ_3 derived from CFDs φ_1, φ_3 and MD ψ in Example 1.1, respectively. Let η be fixed to 0.8. Using Θ and D_m, cRepair finds deterministic fixes for $t_1, t_2 \in D$ *w.r.t.* η as follows.

(1) After initialization (lines 1–6), we have the following: (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 inspected (line 12), we have that $\mathsf{Q}[t_2] = \emptyset$, $\mathsf{P}[t_2] = \{\xi_2\}$, 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 $t_1[\mathsf{city}] := \text{Edi}$. It also upgrades $t_1[\mathsf{city}].\mathsf{cf}$ to 0.8.

(4) When $\xi_3 \in \mathsf{Q}[t_1]$ is used (line 14), it makes a deterministic fix $t_1[\mathsf{phn}] := s_1[\mathsf{tel}]$, and sets $t_1[\mathsf{phn}].\mathsf{cf} = 0.8$. Now we have that $\mathsf{Q}[t_1] = \{\xi_2\}$ 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 setting $t_3[\mathsf{city}] := \text{Ldn}$ and $t_3[\mathsf{city}].\mathsf{cf} := 0.8$. □

Correctness. (1) Algorithm cRepair always terminates. Indeed, each attribute value of a tuple is updated at most once. (2) The algorithm returns all deterministic fixes, since it recursively uses all rules that are applicable to deterministic fixes, until no rules can be applied. Moreover, by the definition of deterministic fixes, the algorithm generates deterministic fixes only. Note that during the process, each tuple is visited at most twice, by considering the following cases.

- Cleaning rules derived from constant CFDs and MDs. When any of these is applied to a tuple $t \in D$, the process is independent of any other tuples in D. Hence, each tuple in D is visited only once.
- Cleaning rules derived from variable CFDs. There are two conditions for a rule derived from a variable CFD $\varphi = (X \to B, t_p)$ to be applicable to a tuple t:
  (i) $t[X]$ is asserted, and
  (ii) there exists a tuple $t' \in D$, such that $t'[X \cup B]$ is asserted and moreover, $t'[B]$ is the only asserted B attribute for all tuples in $\Delta(t[X])$.
  When both (i) and (ii) are satisfied, $t[B]$ will be replaced by $t'[B]$. Hence, vCFDInfer will visit a tuple t at most twice: when $t[X]$ is asserted, and when the first tuple t' is updated by means of t, in which $t'[X \cup B]$ is asserted.

Based on the analysis above, one can readily verify that for deterministic fixes, the order in which rules are applied does not impact the quality of the final result. Moreover, the order has no impact on the efficiency either. Indeed, (1) applying the rules in different orders yield the same set of deterministic fixes, (2) each attribute value of a tuple is updated at most once, and (3) the cost of the update operation is decided by the rule applied and the tuples involved, independent of other rules and tuples.

Similarity checking for MDs. For cleaning rules derived from MDs, *similarity checking* between tuples across relations is required, where *similarity* is typically defined on strings (*i.e.,* attribute values in tuples) employing various string metrics, *e.g.,* Hamming/Edit distance, such that two strings are considered to be *similar* if the difference between their values is within a pre-defined threshold K in the metric. Unfortunately, traditional database indices, *e.g.,* B$^+$-tree and hash tables that are designed for exact

matching, cannot be carried over to our problem. In the following, we shall present techniques for similarity checking that can efficiently deal with either a small or a large threshold K. The techniques can also used to conduct exact matching for its close connection with Hamming/Edit and longest common substring (LCS).

Before we present our techniques, let us first review previous work on suffix-tree based indices based on Hamming/Edit distance [Tsur 2010; Cole et al. 2004]. These indices are efficient for matching tuples within a threshold K. More specifically, [Tsur 2010] can reduce the time for checking the similarity of an input string v and master data D_m from $O(|D_m||v|)$ to $O(m+$ *log log* $n + \#$*-matches)*, where K is a constant, m is the number of tuples in D_m and n is the length of the longest indexed string in D_m. Nevertheless, these indices treat K as a constant, and need exponential time in K. Hence, they are only effective for small K.

When K is relatively large, we develop another technique based on the idea of *blocking*. More specifically, instead of traversing the entire set of tuples in D_m, we use indices to find top-l tuples in D_m that possibly match an input string, where l is a constant determined by users. Blocking is based on the length of LCS, since two strings u and v have a Hamming/Edit distance within K only if the length of their LCS is at least $\mathsf{max}(|u|, |v|)/(K + 1)$, where $\mathsf{max}(|u|, |v|)$ is the length of the longer string between u and v. Note that l is relatively small, since for a given string in real-life, there are often few matching strings in the active domain. In our experimental study, we find that $l \leq 20$ typically suffices. The technique above can reduce the search space from D_m to l tuples, for any given string.

In light of this, we generalize suffix trees as an index for LCS. For each attribute that needs similarity checking, a generalized suffix tree is maintained on those strings in the active domain of the attribute in D_m. Note that each node in the generalized suffix tree corresponds to a common substring s, for which a set of strings that contain s are maintained. To look up a string v of length $|v|$, we can extract the subtree T of the suffix tree that only contains branches related to v, which contains at most $|v|^2$ nodes. We traverse T bottom-up to pick top-l similar strings in terms of the length of the LCS. In this way, we can identify l similar values from D_m in $O(l|v|^2)$ time, and reduce the search space from $|D_m|$ to a constant number l of values.

Our experimental study verifies that these techniques significantly improve the performance (see Section 8).

Complexity. Each tuple t in D is examined at most twice for each CFD in Σ, and is checked at most $|D_m|$ times for each MD. From these it follows that cRepair is in $O(|D||D_m|\mathsf{size}(\Sigma \cup \Gamma))$ time. By employing the optimization methods given above, the time complexity of cRepair can be reduced to $O(|D|\mathsf{size}(\Sigma \cup \Gamma))$.

From the analyses given above Theorem 5.1 follows.

6. RELIABLE FIXES WITH INFORMATION ENTROPY

Deterministic fixes may not exist for some attributes, *e.g.,* those attributes for which the confidences are low or unreliable. To find accurate fixes for such attributes, UniClean looks for evidence from data itself *instead of confidence*, 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 using entropy (Section 6.2). We also present an indexing structure underlining 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.

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. $\mathcal{O}$:= the order of $\Sigma \cup \Gamma$, sorted via their dependency graph;
2. D' := D;
3. **repeat**
4. **for** (i = 1; $i \leq |\Sigma \cup \Gamma|$; i++) **do**
5. ξ := the i-th rule in $\mathcal{O}$;
6. **case** ξ of
7. (1) variable CFD: D' := vCFDReslove(D', ξ, δ_1, δ_2);
8. (2) constant CFD: D' := cCFDReslove(D', ξ, δ_1);
9. (3) MD: D' := MDReslove(D', D_m, ξ, δ_1);
10. **until** there are no changes in D';
11. **return** D'.

Fig. 6. Algorithm eRepair

Entropy. The entropy of a discrete random variable $\mathcal{X}$ with possible values in $\{x_1, \ldots, x_n\}$ is defined as follows [Cover and Thomas 1991; Srivastava and Venkatasubramanian 2010]:

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

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 $\varphi = R(Y \rightarrow B, t_p)$ 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 Fig. 3) 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 follows:

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

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 $p_i = \frac{\mathsf{cnt}_{YB}(\bar{y}, b_i)}{|\Delta(\bar{y})|}$. 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 all $t \in \Delta(\bar{y})$, where b_j is the one with the highest probability, *i.e.,* $\mathsf{cnt}_{YB}(\bar{y}, b_j)$ is 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 $b_i, b_j \in \pi_B(\Delta(\bar{y}))$. Note that 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 next present an algorithm based on entropy, followed by its main procedures and auxiliary structures.

Algorithm. The algorithm, referred to as eRepair, is shown in Fig. 6. 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, algorithm eRepair finds reliable fixes for D and returns a (partially cleaned) database D' in which reliable fixes are marked. The deterministic fixes found earlier by cRepair remain unchanged in the process.

In a nutshell, algorithm eRepair first sorts data 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. More specifically, it 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 then returned in which reliable fixes are marked (line 11).

Procedures. We next present the procedures of eRepair.

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 should be applied before ξ_v. For instance, the dependency graph of the CFDs and MDs given in Example 1.1 is shown in Fig. 7.

Based on the dependency graph G, we sort the rules as follows. (1) Find strongly connected components (SCCs) in G, in linear time [Cormen et al. 2001]. (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 depicted in Fig. 7 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. □

We next present the details of the procedures.

Procedure 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 $\text{cnt}_{YB}(\bar{y}, b)$. 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.

Procedure cCFDReslove. It applies the cleaning 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 for less than δ_1 times, then $t[A]$ is changed to the constant $t_{p_1}[A]$.

Procedure MDReslove. It applies the cleaning 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 mas-

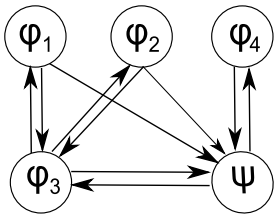


Fig. 7. Example dependency graph

	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

Fig. 8. Example relation of schema R

ter 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.

Example 6.2. Consider an instance of schema $R(\mathsf{ABCEFH})$ shown in Fig. 8, 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 algorithm eRepair will only change $t_4[E]$ to e_1, and marks it as a reliable fix. □

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, where k is a constant. Thus, the algorithm takes $O(\delta_1|D|^2|\Sigma| + \delta_1 k|D|^2\mathsf{size}(\Gamma))$ time in total.

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 $\varphi = R(Y \to B, t_p)$ in Σ_V, we build a structure consisting of a *hash table* and an *AVL tree* [Cormen et al. 2001] 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 $(\mathsf{key}, \mathsf{val})$ into HTab, where $\mathsf{key} = \bar{y}$, 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 $\mathsf{v}_r.\epsilon \geq \mathsf{v}.\epsilon$.

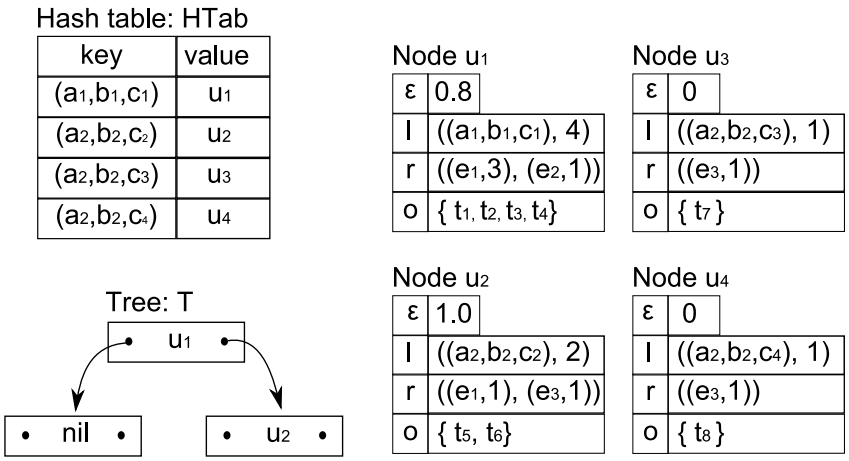


Fig. 9. Example data structure for variable CFDs

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. 8 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. 9. □

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 by capitalizing 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 $O(|D|\text{size}(\Sigma_V))$ space for all CFDs in Σ_V, where $\text{size}(\Sigma_V)$ is the size of Σ_V.

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

7. POSSIBLE FIXES WITH HEURISTICS

The first two modules of system UniClean identify deterministic fixes and reliable fixes for a relation D, using master data D_m and cleaning rules derived from CFDs Σ and MDs Γ. The outcome is a partial repair D' of D in which those fixes are marked. However, D' may still contain errors that are not corrected by deterministic fixes or reliable fixes. In light of these, we need to fix the remaining errors in D' and produce a repair D_r of D such that $D_r \models \Sigma$, $(D_r, D_m) \models \Gamma$, and moreover, D_r preserves the deterministic fixes generated earlier, *i.e.,* such fixes remain unchanged. Observe that the deterministic fixes are assured to be correct up to a confidence level, as shown in Section 5. Hence we decide to keep deterministic fixes unchanged when generating

possible fixes. In contrast, reliable fixes are not guaranteed to be as accurate as deterministic fixes and hence, we opt to allow them to be changed. Nevertheless, we found that only a few reliable fixes were changed in our experiments.

To produce D_r, we adopt the heuristic method of [Cong et al. 2007], which uses the same cost model presented in Section 3.1. We extend the method of [Cong et al. 2007], referred to as hRepair, by (a) supporting matching with master data D_m and cleaning rules derived from MDs, (b) preserving the deterministic fixes generated earlier, and (c) keeping reliable fixes generated earlier as many as possible.

The key idea of heuristic fixes is the usage of equivalent classes [Cong et al. 2007; Bohannon et al. 2005]. An *equivalence class* consists of pairs of the form (t, A), where t identifies a tuple in which A is an attribute. In a database D, each tuple t and each attribute A in t have an associated equivalence class, denoted by $\mathsf{eq}(t, A)$. In a repair a unique *target value* is assigned to each equivalence class E, denoted by $\mathsf{targ}(E)$. That is, for all $(t, A) \in E$, $t[A]$ has the same value $\mathsf{targ}(E)$. The target value $\mathsf{targ}(E)$ can be either '_', a constant a, or null, where '_' indicates that $\mathsf{targ}(E)$ is not yet fixed, and null means that $\mathsf{targ}(E)$ is uncertain due to conflict. To resolve CFD violations $\mathsf{targ}(E)$ may be "upgraded" from '_' to a constant a, or from a to null, but not the other way around. In particular, $\mathsf{targ}(E)$ is *not* allowed to be changed from one constant to another. Intuitively, we resolve CFD violations by *merging* or *upgrading* the target values of equivalence classes. The process to handle MD violations is along the same lines, by incorporating constant values from the master relation D_m.

Following [Cong et al. 2007], we adopt the *simple* semantics of the SQL standard [International Standard ISO/IEC 9075-2:2003(E) 2003] for null: $t_1[X] = t_2[X]$ evaluates to true if *either one* of them contains null. In contrast, when matching a data tuple t and a pattern tuple t_p, $t[X] \asymp t_p[X]$ is false if $t[X]$ contains null, *i.e.,* CFDs only apply to those tuples that precisely match a pattern tuple, which does not contain null. When matching a data tuple with a master tuple using MDs, we also adopt the same approach as for CFDs.

Observe the following. (a) At each step of the heuristic process to generate possible fixes, either the total number N of equivalences classes is reduced or the number H of those classes that are assigned a constant or null is increased. Let k be the number of (t, A) pairs in D. Since $N \leq k$ and $H \leq 3 \cdot k$ (the target value of $\mathsf{eq}(t, A)$ can only be '_', a constant, or null), the data cleaning process necessarily terminates. Moreover, since the process proceeds until no more dirty tuples exist, it always finds a repair of D that satisfies all CFDs and MDs. (b) The repairing process still terminates with a repair if we keep the deterministic fixes unchanged, by the definition of deterministic fixes and by capitalizing on null when resolving conflicts.

Putting these together, one can verify the following.

Corollary 7.1: *Given a set Σ of CFDs, a set Γ of MDs, master data D_m, and a partial repair D' of a relation D with deterministic fixes and reliable fixes,* hRepair *always finds a repair D_r of D such that $D_r \models \Sigma$, $(D_r, D_m) \models \Gamma$, and D_r preserves all the deterministic fixes in D'.* □

Example 7.2. Recall the relation D, CFDs φ_1–φ_4 and the MD ψ from Example 1.1. As shown in Examples 5.2 and 6.2, algorithms cRepair and eRepair identify several fixes for D. However, the data still has errors, *e.g.,* the tuple t_3 in D does not satisfy the CFD φ_4 even after $t_3[\mathsf{city}]$ is fixed. To this end we find possible fixes using hRepair: (a) $t_3[\mathsf{FN}]$:= Robert by applying the cleaning rule derived from φ_4, (b) $t_3[\mathsf{phn}]$:= 3887644 by matching the master tuple s_2 with the rule derived from the MD ψ, and (c) $t_4[\mathsf{St}, \mathsf{post}]$:= $t_3[\mathsf{St}, \mathsf{post}]$ with the rule derived from φ_3. After these steps we get a repair of D

that satisfies both the CFDs and MDs, and moreover, retains the deterministic fixes generated earlier. □

8. EXPERIMENTAL STUDY

We next present an experimental study of UniClean, which unifies matching and repairing operations. Using both real-life data and synthetic 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 dirty data and the size of master data.

Experimental Setting. We used two real-life data sets and one synthetic data set.

(1) HOSP *data* was taken from US Department of Health & Human Services (http://www.hospitalcompare.hhs.gov/). It has $100K$ records with 19 attributes. We manually designed 23 CFDs and 3 MDs for HOSP, 26 in total.

(2) DBLP *data* was extracted from DBLP Bibliography (http://www.informatik.uni-trier.de/~ley/db/). It consists of $400K$ tuples, each with 12 attributes. We manually designed 7 CFDs and 3 MDs for DBLP, 10 in total. Note that DBLP has tried to identify and distinguish authors with the same name. When people with the same name are identified by DBLP to be different persons, we treat them as different entries. Otherwise, we do not differentiate them.

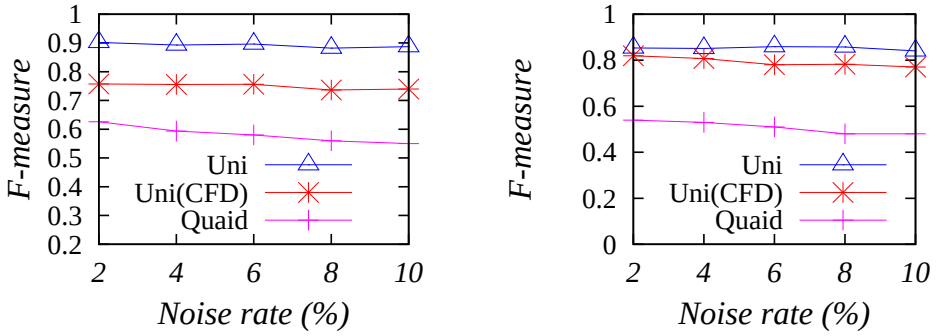
(3) TPC-H *data* was generated from TPC-H benchmark [Transaction Processing Performance Council 2013] by joining all tables together into a single table. It consists of $100K$ tuples, each with 58 attributes. We manually designed 55 FDs, and controlled the number of CFDs and MDs by adding pattern to the FDs for the experiments for efficiency. In total 55 CFDs and 10 MDs were used by default.

(4) Dirty datasets were produced by introducing noises to data from the the 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%. The sources are used as ground truth for effectiveness evaluation.

Master data for (1), (2) and (3) was carefully selected from the same data sources so that they were verified to be correct and consistent *w.r.t.* the designed rules. The master data are separated from the data sources that we used for introducing dirty datasets. For (1), we examined the source data, and found that all the tuples were consistent with the CFDs and MDs we designed. Thus, we treated all the 100K tuples before noises were introduced as candidate master data. For (2), we randomly selected 100K tuples from the source data after removing those tuples that were inconsistent with the CFDs or MDs as candidate master data. For (3), the master data was the data generated by the generator without introducing noises, which also satisfied the CFDs and MDs. In all the experiments, we used 20K tuples from candidate master data as D_m by default.

For all the datasets above, the CFDs and MDs were designed manually. In fact, for datasets where sufficient domain expertise is not available, both CFDs and MDs can be automatically discovered from data via profiling algorithms (*e.g.,* [Chiang and Miller 2008; Song and Chen 2009]).

Algorithms. We implemented the following algorithms, all in Python: (a) algorithms cRepair, eRepair and hRepair (an extension of algorithm in [Cong et al. 2007]) in



(a) HOSP: accuracy of repairing (b) DBLP: accuracy of repairing

Fig. 10. Experiment 1: Matching helps repairing

UniClean; (b) the sorted neighborhood method of [Hernandez and Stolfo 1998], denoted by SortN, for record matching based on MDs only; and (c) the heuristic repairing algorithm of [Cong et al. 2007], 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. 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. On the other hand, 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. The F-measure is defined as:

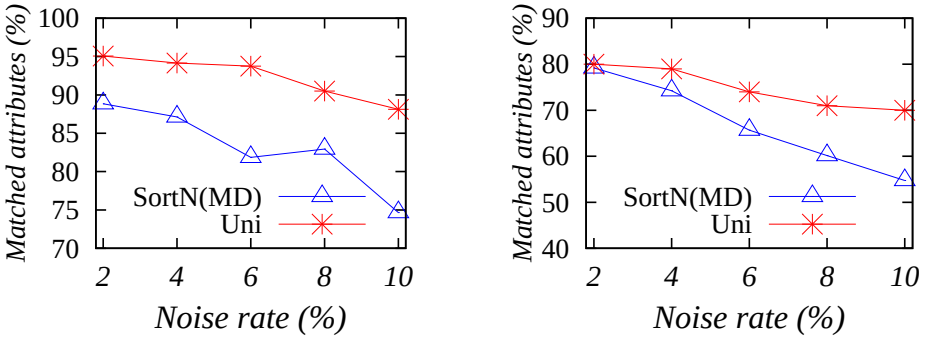
$$\text{F-measure} = 2 \cdot (\text{precision} \cdot \text{recall})/(\text{precision} + \text{recall}).$$

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. Note that TPC-H is a synthetic data, and we used it mainly for efficiency study. In other words, for effectiveness study, we only used real-life data, HOSP and DBLP.

We next report our experimental 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.



(a) HOSP: accuracy of matching (b) DBLP: accuracy of matching

Fig. 11. Experiment 2: Repairing helps matching

To favor Uni(CFD) and quaid, which use CFDs only, we focused on the impact of various noise rates.

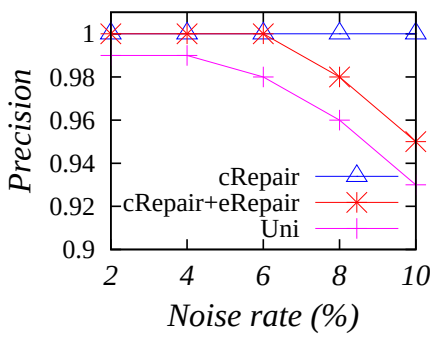
The results on HOSP data and DBLP data are reported in Figures 10(a) and 10(b), respectively, which tell us the following. (1) Uni clearly outperforms Uni(CFD) and quaid by up to 15% and 30%, respectively. This verifies that matching indeed helps repairing. (2) The F-measure typically decreases when 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 with CFDs only, our method 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 evaluated 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 was very small; hence we do not report it here.

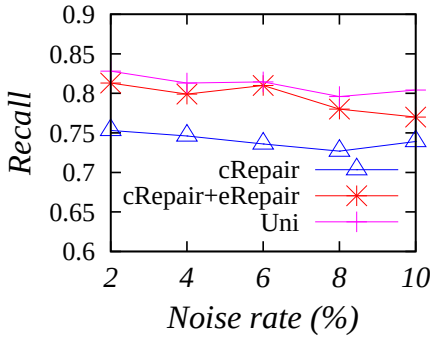
The results are reported in Figures 11(a) and 11(b) for HOSP and DBLP, respectively. We find the following. (a) Uni outperforms SortN(MD) by up to 15%, verifying that repairing indeed helps matching. (b) The F-measure decreases when the noise rate increases for both approaches. However, Uni with repairing is less sensitive to noi%, which is consistent with our observation in the last set of experiments.

Exp-3: Accuracy of deterministic fixes and reliable fixes. We now 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 12(a)–12(d).

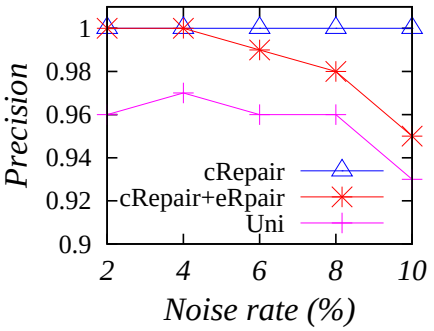
The results tell us the following: (a) Deterministic fixes have the highest precision, and are insensitive to the noise rate noi%. However, their recall is low, since cRepair is "picky": it only generates fixes with asserted attributes. (b) Fixes generated by Uni have the lowest precision, but the highest recall, as expected. Furthermore, 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. Furthermore, their precision is also sensitive to noi%. From these we can see that the precision of reliable



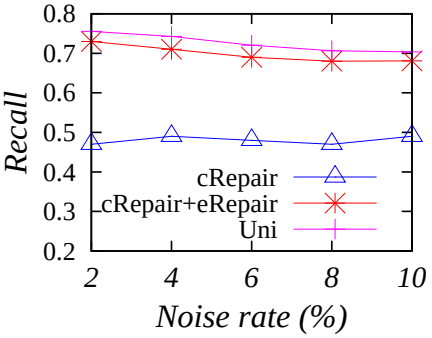
(a) HOSP: precision (3 phases)



(b) HOSP: recall (3 phases)

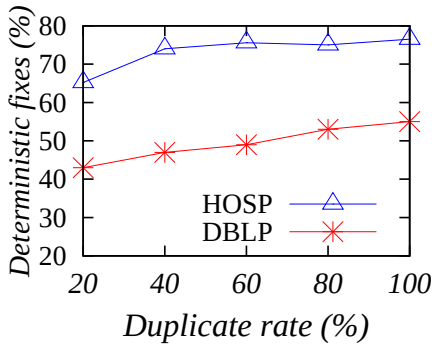


(c) DBLP: precision (3 phases)

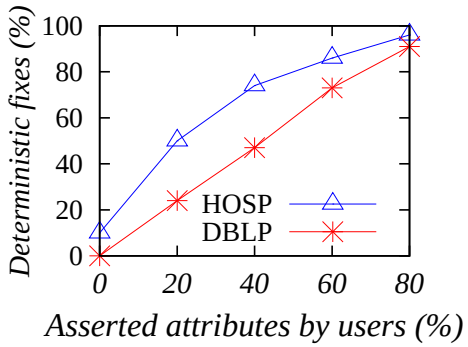


(d) DBLP: recall (3 phases)

Fig. 12. Experiment 3: Accuracy of deterministic fixes and reliable fixes



(a) Deterministic fixes on dup%



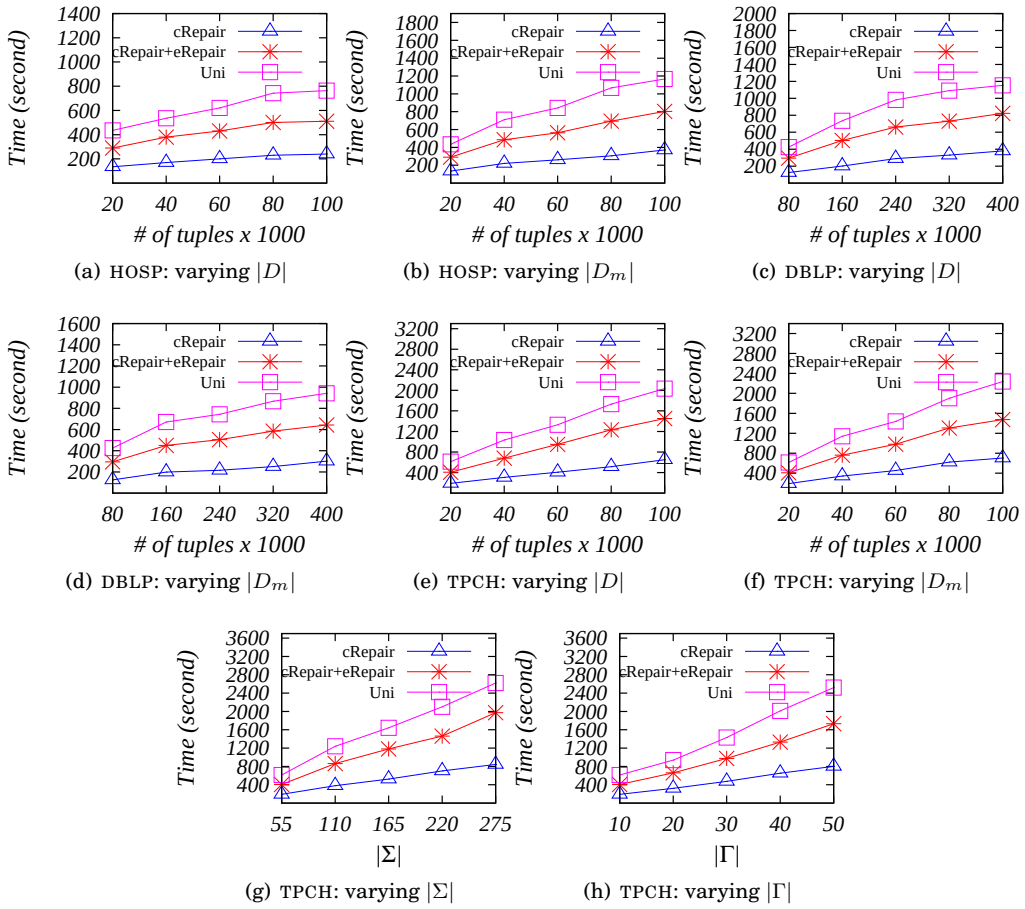
(b) Deterministic fixes on asr%

Fig. 13. Experiment 4: Impact of dup% and asr% on deterministic fixes

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 13(a) shows the results. We find that the larger dup% is, the more deterministic fixes are found, as expected.



(a) HOSP: varying $|D|$ (b) HOSP: varying $|D_m|$ (c) DBLP: varying $|D|$

(d) DBLP: varying $|D_m|$ (e) TPCH: varying $|D|$ (f) TPCH: varying $|D_m|$

(g) TPCH: varying $|\Sigma|$ (h) TPCH: varying $|\Gamma|$

Fig. 14. Experiment 5: Scalability

Fixing dup% $=$ 40%, we varied asr% from 0% to 80%. The results are shown in Fig. 13(b), 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 set of experiments evaluated the scalability of Uni with the size $|D|$ of dirty data, the size $|D_m|$ of master data, the number of CFDs $|\Sigma|$ and the number of MDs $|\Gamma|$. We fixed noi% = 6% and dup% = 40% in these experiments. The results are reported in Figures 14(a) and 14(b) for HOSP, 14(c) and 14(d) for DBLP, and 14(e), 14(f), 14(g) and 14(h) for TPC-H.

Figures 14(a) (resp. 14(b)) shows three curves for HOSP data: the running time of cRepair, cRepair + eRepair and cRepair + eRepair + hRepair (total time of Uni) by fixing $|D_m| = 20K$ (resp. $|D| = 20K$) and varying $|D|$ (resp. $|D_m|$) from $20K$ to $100K$. The results show that with the suffix tree blocking, Uni scales reasonably well with $|D|$ and $|D_m|$. Each step (cRepair, eRepair or hRepair) of Uni takes comparable time. Without the suffix tree blocking, it scales much worse. Indeed, when $|D|$ or $|D_m|$ is $20K$, it took more than 5 hours; thus we omit the curve in the figures. These results verify the

effectiveness of our indexing structures and optimization techniques. The results are consistent for DBLP (resp. TPCH) data, as shown in Figures 14(c) (resp. 14(e)) and 14(d) (resp. 14(f)).

Figures 14(g) (resp. 14(h)) reports the running time of cRepair, cRepair + eRepair and cRepair + eRepair + hRepair on TPCH when varying the number of CFDs $|\Sigma|$ (resp. MDs $|\Gamma|$) from 55 to 275 (resp. 10 to 50). One can see that Uni scales well with both $|\Sigma|$ and $|\Gamma|$.

In fact Uni scales much better than quaid [Cong et al. 2007]: quaid took more than 10 hours when $|D|$ is $80K$ on HOSP, while it took Uni about 11 minutes.

Summary. From the experimental results 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, the size of master data, the number of CFDs and the number of MDs.

9. 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 associated with data cleaning based on both matching and repairing. We have also proposed deterministic fixes and reliable fixes, and developed 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 real-life datasets. We are also exploring optimization techniques to improve the efficiency of our algorithms. Another topic for future work is to study cleaning of multiple relations of which the consistency is specified by constraints across relations, *e.g.,* (conditional) inclusion dependencies [Bravo et al. 2007], an issue more intriguing than cleaning a single relation.

Acknowledgments. Fan is supported in part by 973 Programs 2012CB316200, 2014CB340302, NSFC 61133002, China, and EPSRC EP/J015377/1, UK. Ma is supported in part by NSFC grant 61322207, NGFR 973 grant 2014CB340304 and MOST grant 2012BAH46B04. Fan and Ma are also supported in part by Guangdong Innovative Research Team Program 2011D005 and Shenzhen Peacock Program 1105100030834361 of China.

REFERENCES

ABITEBOUL, S., HULL, R., AND VIANU, V. 1995. *Foundations of Databases*. Addison-Wesley.

AIKEN, A., KOZEN, D., VARDI, M. Y., AND WIMMERS, E. L. 1993. The complexity of set constraints. In *CSL*.

ARASU, A., RE, C., AND SUCIU, D. 2009. Large-scale deduplication with constraints using Dedupalog. In *ICDE*.

ARENAS, M., BERTOSSI, L. E., AND CHOMICKI, J. 2003. Answer sets for consistent query answering in inconsistent databases. *TPLP 3,* 4-5, 393–424.

BAYARDO, R. J., MA, Y., AND SRIKANT, R. 2007. Scaling up all pairs similarity search. In *WWW*.

BERTOSSI, L. E., KOLAHI, S., AND LAKSHMANAN, L. V. S. 2011. Data cleaning and query answering with matching dependencies and matching functions. In *ICDT*.

BESKALES, G., SOLIMAN, M. A., ILYAS, I. F., AND BEN-DAVID, S. 2009. Modeling and querying possible repairs in duplicate detection. *PVLDB 2,* 1, 598–609.

BOHANNON, P., FAN, W., FLASTER, M., AND RASTOGI, R. 2005. A cost-based model and effective heuristic for repairing constraints by value modification. In *SIGMOD*.

BOLLEMAN, J., GATEAU, A., GEHANT, S., AND REDASCHI, N. 2010. Provenance and evidence in uniprotkb. In *SWAT4LS*.

BRAVO, L., FAN, W., AND MA, S. 2007. Extending dependencies with conditions. In *VLDB*.

CAO, Y., CHEN, Z., ZHU, J., YUE, P., LIN, C.-Y., AND YU, Y. 2011. Leveraging unlabeled data to scale blocking for record linkage. In *IJCAI*.

CHAUDHURI, S., GANJAM, K., GANTI, V., AND MOTWANI, R. 2003. Robust and efficient fuzzy match for online data cleaning. In *SIGMOD*.

CHIANG, F. AND MILLER, R. 2008. Discovering data quality rules. In *VLDB*.

CHRISTEN, P. 2012. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE Trans. Knowl. Data Eng. 24,* 9, 1537–1555.

COLE, R., GOTTLIEB, L.-A., AND LEWENSTEIN, M. 2004. Dictionary matching and indexing with errors and don't cares. In *STOC*.

CONG, G., FAN, W., GEERTS, F., JIA, X., AND MA, S. 2007. Improving data quality: Consistency and accuracy. In *VLDB*.

CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., AND STEIN, C. 2001. *Introduction to Algorithms*. The MIT Press.

COVER, T. M. AND THOMAS, J. A. 1991. *Elements of Information Theory*. Wiley-Interscience.

DONG, X., BERTI-EQUILLE, L., HU, Y., AND SRIVASTAVA, D. 2010. Global detection of complex copying relationships between sources. *PVLDB 3,* 1, 1358–1369.

DONG, X., HALEVY, A. Y., AND MADHAVAN, J. 2005. Reference reconciliation in complex information spaces. In *SIGMOD*.

ECKERSON, W. W. 2002. Data Quality and the Bottom Line: Achieving Business Success through a Commitment to High Quality Data. Tech. rep., The Data Warehousing Institute.

ELMAGARMID, A. K., IPEIROTIS, P. G., AND VERYKIOS, V. S. 2007. Duplicate record detection: A survey. *TKDE 19,* 1, 1–16.

FAN, W. 2008. Dependencies revisited for improving data quality. In *PODS*.

FAN, W., GAO, H., JIA, X., LI, J., AND MA, S. 2011a. Dynamic constraints for record matching. *VLDB J. 20,* 4, 495–520.

FAN, W., GEERTS, F., JIA, X., AND KEMENTSIETSIDIS, A. 2008. Conditional functional dependencies for capturing data inconsistencies. *TODS 33,* 1.

FAN, W., GEERTS, F., LI, J., AND XIONG, M. 2011b. Discovering conditional functional dependencies. *TKDE 23,* 5, 683–698.

FAN, W., LI, J., MA, S., TANG, N., AND YU, W. 2010. Towards certain fixes with editing rules and master data. *PVLDB 3,* 1, 173–184.

FAN, W., LI, J., MA, S., TANG, N., AND YU, W. 2011c. Interaction between record matching and data repairing. In *SIGMOD*.

FELLEGI, I. AND HOLT, D. 1976. A systematic approach to automatic edit and imputation. *J. American Statistical Association 71,* 353, 17–35.

GAREY, M. AND JOHNSON, D. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company.

GARTNER. 2007. Forecast: Data quality tools, worldwide, 2006-2011. Tech. rep., Gartner.

GUO, S., DONG, X., SRIVASTAVA, D., AND ZAJAC, R. 2010. Record linkage with uniqueness constraints and erroneous values. *PVLDB 3,* 1, 417–428.

HAMMING, R. W. 1950. Error detecting and error correcting codes. *Bell System Technical Journal 29,* 2, 147–160.

HERNANDEZ, M. A. AND STOLFO, S. 1998. Real-World Data is Dirty: Data Cleansing and the Merge/Purge Problem. *Data Mining and Knowledge Discovery 2,* 1, 9–37.

HERZOG, T. N., SCHEUREN, F. J., AND WINKLER, W. E. 2009. *Data Quality and Record Linkage Techniques*. Springer.

INTERNATIONAL STANDARD ISO/IEC 9075-2:2003(E). 2003. Information technology: Database languages, SQL Part 2 (Foundation, 2nd edition).

KLIR, G. J. AND FOLGER, T. A. 1988. *Fuzzy sets, uncertainty, and information*. Englewood Cliffs, N.J: Prentice Hall.

LOSHIN, D. 2009. *Master Data Management*. Knowledge Integrity, Inc.

MAYFIELD, C., NEVILLE, J., AND PRABHAKAR, S. 2010. ERACER: a database approach for statistical inference and data cleaning. In *SIGMOD*.

NAUMANN, F., BILKE, A., BLEIHOLDER, J., AND WEIS, M. 2006. Data fusion in three steps: Resolving schema, tuple, and value inconsistencies. *IEEE Data Eng. Bull. 29,* 2, 21–31.

OTTO, B. AND WEBER, K. 2009. From health checks to the seven sisters: The data quality journey at BT. BT TR-BE HSG/CC CDQ/8.

RAHM, E. AND DO, H. H. 2000. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull. 23,* 4, 3–13.

REDMAN, T. 1998. The impact of poor data quality on the typical enterprise. *Commun. ACM 2*, 79–82.

SAVITCH, W. J. 1970. Relationships between Nondeterministic and Deterministic Tape Complexities. *JCSS 4*, 177–192.

SONG, S. AND CHEN, L. 2009. Discovering matching dependencies. In *CIKM*.

SRIVASTAVA, D. AND VENKATASUBRAMANIAN, S. 2010. Information theory for data management. In *SIGMOD*.

TRANSACTION PROCESSING PERFORMANCE COUNCIL. 2001-2013. TPC-H benchmark. `http://www.tpc.org`.

TSUR, D. 2010. Fast index for approximate string matching. *J. of Discrete Algorithms 8,* 4, 339–345.

WANG, J., LI, G., AND FENG, J. 2010. Trie-join: Efficient trie-based string similarity joins with edit-distance constraints. *PVLDB 3,* 1, 1219–1230.

WANG, J., LI, G., AND FENG, J. 2011. Fast-join: An efficient method for fuzzy token matching based string similarity join. In *ICDE*.

WEGENER, I. AND PRUIM, R. 2005. *Complexity Theory: Exploring the Limits of Efficient Algorithms*. Springer.

WEIS, M. AND NAUMANN, F. 2005. Dogmatix tracks down duplicates in XML. In *SIGMOD*.

WHANG, S. E., BENJELLOUN, O., AND GARCIA-MOLINA, H. 2009. Generic entity resolution with negative rules. *VLDB J. 18,* 6, 1261–1277.

WIJSEN, J. 2005. Database repairing using updates. *TODS 30,* 3, 722–768.

XIAO, C., WANG, W., AND LIN, X. 2008. Ed-join: an efficient algorithm for similarity joins with edit distance constraints. *PVLDB 1,* 1, 933–944.

XIAO, C., WANG, W., LIN, X., YU, J. X., AND WANG, G. 2011. Efficient similarity joins for near-duplicate detection. *ACM Trans. Database Syst. 36,* 3, 15.

YAKOUT, M., ELMAGARMID, A. K., NEVILLE, J., AND OUZZANI, M. 2010. GDR: a system for guided data repair. In *SIGMOD*.

ZIV, J. AND LEMPEL, A. 1978. Compression of individual sequences via variable-rate coding. *IEEE TIT 24,* 5, 530–536.