Determining the relative accuracy of attributes

Determining the Relative Accuracy of Attributes

Yang Cao[1,2] Wenfei Fan[1,2] Wenyuan Yu[1]
[1]School of Informatics, University of Edinburgh
[2]Big Data Research Center and SKLSDE Lab, Beihang University
{yang.cao@, wenfei@inf., wenyuan.yu@}ed.ac.uk

ABSTRACT

The *relative accuracy problem* is to determine, given tuples t_1 and t_2 that refer to the same entity e, whether $t_1[A]$ is more accurate than $t_2[A]$, *i.e.,* $t_1[A]$ is *closer to the true value* of the A attribute of e than $t_2[A]$. This has been a longstanding issue for data quality, and is challenging when the true values of e are unknown. This paper proposes a model for determining relative accuracy. (1) We introduce a class of accuracy rules and an inference system with a chase procedure, to deduce relative accuracy. (2) We identify and study several fundamental problems for relative accuracy. Given a set I_e of tuples pertaining to the same entity e and a set of accuracy rules, these problems are to decide whether the chase process terminates, is Church-Rosser, and leads to a unique *target tuple* t_e composed of the most accurate values from I_e for all the attributes of e. (3) We propose a framework for inferring accurate values with user interaction. (4) We provide algorithms underlying the framework, to find the unique target tuple t_e whenever possible; when there is no enough information to decide a complete t_e, we compute top-k candidate targets based on a preference model. (5) Using real-life and synthetic data, we experimentally verify the effectiveness and efficiency of our method.

Categories and Subject Descriptors

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

Keywords

data accuracy, data cleaning

1. INTRODUCTION

One of the central issues of data quality is *data accuracy*. Given a set I_e of tuples pertaining to the same entity e, it aims to find the most accurate values for e. More specifically, it is to compute a tuple t_e, referred to the *target tuple for e from I_e*, such that for each attribute A of e, $t_e[A]$ is a value in I_e that is closest to the *true A-value* of e. The need for studying this is evident in *e.g.,* decision making [20], information systems [10] and data quality management [18].

Although important, data accuracy has not been well studied. Prior work on data quality has mostly focused on other issues such as data consistency [2, 12]. Consistency refers to the validity and integrity of the data. While there is an intimate connection between data accuracy and consistency, the two are *quite different*. Indeed, a database D may be consistent, but the values in D may still be inaccurate.

Example 1: Consider relation stat given in Table 1, which collects performance statistics of Michael Jordan (Fig. 1) in the season of 1994-95, when Michael played for a baseball team Birmingham Barons in the Southern League (SL) in 1994, followed by his return to NBA, playing 27 games for Chicago Bulls in 1995. Each tuple in stat specifies the name (FN, MN, LN), performance (total points totalPts after rnds rounds played), jersey number J#, league, team and arena.

We want to find the target tuple for Michael from stat, consisting of the most accurate values for all the attributes at the end of 1994-95 NBA season. However, there are multiple values in stat for some attributes. For instance, we do not know whether 45 is more accurate than 23 for his J#.

The stat data is *consistent*. Indeed, constraints specifying its consistency include (a) functional dependency (FD [1]): [FN, MN, LN, league, rnds $\rightarrow$ totalPts], *i.e.,* player, rnds and league uniquely determine totalPts, and (b) conditional functional dependency (CFD [13]): [team = "Chicago Bulls" $\rightarrow$ arena = "United Center"], asserting that if team is Chicago Bulls, then arena must be United Center. While all tuples in stat satisfy these constraints and are hence *consistent*, most of the data values in stat are, however, *not accurate*. □

Not all is lost. An enterprise typically maintains *master data* [25], a single repository of high-quality data that provides various applications with a synchronized, consistent view of its core business entities. Leveraging master data, one is able to identify accurate values for *some* attributes. For example, a master relation nba is given in Table 2, in which a tuple specifies the FN, LN, league, season, and team of an NBA player. Then tuple s_1 in Table 2 tells us that in the 1994-95 season, Michael played for Chicago Bulls in NBA. Thus t_1 is *more accurate* than t_4 in *attribute* league (resp. team), denoted by $t_6 \prec_{\mathsf{league}} t_1$ (resp. $t_4 \prec_{\mathsf{team}} t_1$).

It is more challenging, however, to determine the *relative accuracy* of those attributes *in the absence of complete master data*. That is, given tuples t_1, t_2 and an attribute A, we want to know whether $t_1 \prec_A t_2$ when A is *not* covered by master data, such as J# in relation stat of Table 1. This is hard, but not hopeless. From the semantics of the data, one can derive accuracy rules (ARs), which tell us whether one tuple is more accurate than another in certain attributes.

Example 2: An analysis of the semantics of the stat data yields the ARs given in Table 1. Based on these rules, we can deduce relative accuracy as follows.

(1) We know that in a season, the number of rounds (rnds) monotonically increases (up to an bound). Hence for tuples t and t' referring to the same league, if $t[\mathsf{rnds}] < t'[\mathsf{rnds}]$, then $t \prec_{\mathsf{rnds}} t'$, *i.e.,* $t'[\mathsf{rnds}]$ is more current (and thus more accu-

SIGMOD'13, June 22–27, 2013, New York, New York, USA.

	FN	MN	LN	rnds	totalPts	J#	league	team	arena
t_1:	MJ	null	null	16	424	45	NBA	Chicago	Chicago Stadium
t_2:	Michael	null	Jordan	27	772	23	NBA	Chicago Bulls	United Center
t_3:	Michael	null	Jordan	1	19	45	NBA	Chicago Bulls	United Center
t_4:	Michael	Jeffrey	Jordan	127	51	45	SL	Birmingham Barons	Regions Park

Table 1: Entity instance stat for Michael Jordon in the 1994-95 season

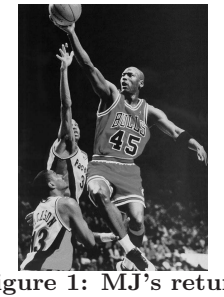


Figure 1: MJ's return

	FN	LN	league	season	team
s_1:	Michael	Jordan	NBA	1994-95	Chicago Bulls
s_2:	Michael	Jordan	NBA	2001-02	Washington Wizards

Table 2: Master data nba

φ_1: $\forall t_1, t_2 \in \mathsf{stat}$ $(t_1[\mathsf{league}] = t_2[\mathsf{league}] \wedge t_1[\mathsf{rnds}] < t_2[\mathsf{rnds}] \rightarrow t_1 \preceq_{\mathsf{rnds}} t_2)$
φ_2 : $\forall t_1, t_2 \in \mathsf{stat}$ $(t_1 \prec_{\mathsf{rnds}} t_2 \rightarrow t_1 \preceq_{\mathsf{J\#}} t_2)$
φ_3: $\forall t_1, t_2 \in \mathsf{stat}$ $(t_1 \prec_{\mathsf{rnds}} t_2 \rightarrow t_1 \preceq_{\mathsf{totalPts}} t_2)$
φ_4 : $\forall t_1, t_2 \in \mathsf{stat}$ $(t_1 \prec_{\mathsf{league}} t_2 \rightarrow t_1 \preceq_{\mathsf{rnds}} t_2)$
φ_5 : $\forall t_1, t_2 \in \mathsf{stat}$ $(t_1 \prec_{\mathsf{MN}} t_2 \rightarrow t_1 \preceq_{\mathsf{FN}} t_2)$
φ_6 : $\forall t_m \in \mathsf{nba}(t_m[\mathsf{FN}, \mathsf{LN}] = t_e[\mathsf{FN}, \mathsf{LN}] \wedge t_m[\mathsf{season}] =$ "1994-95" $\rightarrow t_e[\mathsf{league}, \mathsf{team}] = t_m[\mathsf{league}, \mathsf{team}])$

Table 3: Accuracy rules (ARs)

rate) than t[rnds]. This is expressed as rule φ_1 in Table 1. From φ_1 we can deduce that $t_i \prec_{\mathsf{rnds}} t_2$ for $i \in [1, 3]$.

(2) For tuples t and t', if t' is more accurate than t in rnds, then $t \preceq_{\mathsf{J\#}} t'$, denoting either $t[\mathsf{J\#}] = t'[\mathsf{J\#}]$ or $t \prec_{\mathsf{J\#}} t'$; similarly for totalPts. That is, if t' is more accurate (current) than t in attribute rnds, then so are its *correlated attributes* $t'[\mathsf{J\#}]$ and $t'[\mathsf{totalPts}]$. These are expressed as rules φ_2 and φ_3 in Table 1, respectively. From these ARs and (1) we find that $t_2[\mathsf{totalPts}] = 772$ and $t_2[\mathsf{J\#}] = 23$ are more accurate than $t_i[\mathsf{totalPts}]$ and $t_i[\mathsf{J\#}]$, respectively, for $i \in [1, 3]$.

(3) We know that Michael ended up in NBA in the 1994-95 season. Moreover, if t' is more accurate than t in league, then so is t' in attributes rnds, totalPts, J# and arena. These can also be expressed as ARs, *e.g.,* φ_4. These tell us that t_i is more accurate than t_4 in these attributes, for $i \in [1, 3]$.

(4) For tuples t and t', if $t[\mathsf{A}]$ is null but $t'[\mathsf{A}]$ is not, then t' is more accurate than t in attribute A. This can also be expressed as an AR (not shown in Table 1). Moreover, if t' is more accurate than t in MN, then so is t' in the *correlated attribute* FN, as $t'[\mathsf{MN}]$ and $t'[\mathsf{FN}]$ typically *come together*. This is expressed as φ_5 in Table 1. These tell us that $t_4[\mathsf{MN}]$="Jeffrey" is most accurate in MN and $t_1 \prec_{\mathsf{FN}} t_4$.

(5) As remarked earlier, we can use master data to find the most accurate values for certain attributes. This is shown as AR φ_6 in Table 1. It asserts that if there exists a master tuple $t_m \in \mathsf{nba}$ such that $t_m[\mathsf{FN}, \mathsf{LN}] = t_e[\mathsf{FN}, \mathsf{LN}]$ and $t_m[\mathsf{season}] =$ "1994-95", then $t_e[\mathsf{league}, \mathsf{team}]$ should take the value of $t_m[\mathsf{league}, \mathsf{team}]$. Here t_e is the target tuple in which attributes $t_e[\mathsf{FN}, \mathsf{LN}]$ already find their most accurate values.

Putting these together, we can deduce the relative accuracy of attributes and better still, a large part of the target tuple t_e. Indeed, the values of t_e in FN, MN, LN, rnds, totalPts, league, team are found to be Michael, Jeffrey, Jordan, 27, 772, 23, NBA, and Chicago Bulls, respectively. □

This example tells us that *in the absence of true values* of an entity, one can still determine a large part of the target tuple, by taking together ARs and available master data. This, however, requires an inference system and efficient algorithms for deducing the relative accuracy of attributes.

Contributions. We make a first attempt to give a formal treatment of relative accuracy, from theory to practice.

(1) We propose a model for determining relative accuracy (Section 2). We introduce (a) *accuracy rules* (ARs) defined in terms of partial orders; and (b) a chase-like procedure [1] that, given a set I_e of tuples pertaining to the same entity e, a set Σ of ARs on I_e and (partial) master data I_m, infers relative accuracy and a target tuple by applying the ARs.

(2) We identify fundamental problems for relative accuracy (Section 3). Given I_e, Σ and I_m, these problems are to decide (a) whether the chase process on I_e terminates by applying Σ and I_m? (b) Whether do all the chase sequences lead to a unique target tuple t_e for e from I_e, *i.e.,* Church-Rosser [1], no matter in what orders the rules are applied? (c) If t_e is incomplete, *i.e.,* some of its attributes have the null value, can we make t_e complete while observing the ARs in Σ? We show that the chase process always terminates, the Church-Rosser property can be decided in $O((|I_e|^2 + |I_m|)|\Sigma|)$ time, whereas the last problem is NP-complete.

(3) We present a framework for deducing target tuples (Section 4). Given I_e, Σ and I_m, the framework checks whether the chase on I_e with Σ and I_m is Church-Rosser. If so, it *automatically deduces* as many accurate attribute values for t_e as possible. If t_e is incomplete, it computes top-k candidate targets based on a preference model. The users may check the candidate tuples, revise t_e, I_e and Σ, and invoke the process again until a satisfactory target tuple is found.

(4) We provide effective algorithms underlying the framework (Sections 5 and 6). We give an algorithm for deciding whether the chase is Church-Rosser given I_e, Σ and I_m, and deducing accurate attributes for target tuples. We also develop three algorithms for finding top-k candidate target tuples, with the *early termination* property without inspecting all possible tuples. In particular, one of the algorithms does not require ranked lists as input, and is *instance optimal w.r.t.* the number of visits to the data [11].

(5) We experimentally verify the effectiveness and efficiency of our method, using real-life and synthetic data (Section 7). We find that our approach is effective: for the real-life data, accurate values are automatically deduced for at least 73% of the attributes without user interaction, in 10 milliseconds (ms); moreover, at most 3-4 rounds of user interaction are needed to find complete target tuples. Our algorithms scale well with the sizes of entity instances, master data and ARs. We also evaluate our method for truth discovery, vs. prior approaches [8, 14]. We show that our model can accommodate trust in data sources [8] as well as data currency and consistency [14]. Even for truth discovery, our method performs as well as [8, 14] in their settings, or even better.

The analysis of relative accuracy helps us deduce the true value of an entity e. Even when we do not have enough

information to decide the true values of all the attributes of e, partial true values deduced may still *enhance our trust* in values *critical to* decision making. In addition, relative accuracy helps us reduce search space and improve accuracy in data repairing, which is computationally costly [12]. Indeed, if $t_1 \prec_A t_2$ is deduced, we know that $t_1[A]$ is a better candidate than $t_2[A]$ for the A-attribute of these tuples. As will be seen shortly, relative accuracy is deduced via logical inference from data semantics. The inference can be *combined with* probabilistic analysis techniques (see below), to facilitate truth discovery, data repairing and decision making.

We contend that ARs, master data and inference algorithms yield a promising approach to determining relative accuracy. As shown in Example 2, while master data is helpful, it is *not* a must for the analysis of relative accuracy. In the absence of complete master data, we can still deduce true values for critical attributes based on accuracy rules and inference, as will be verified by our experimental study.

Related works. There has been a host of work on data quality issues such as data consistency, data currency, information completeness and entity resolution (see [2, 12] for recent surveys). While data accuracy has long been advocated [2,16,23], the prior work has mostly focused on metrics for accuracy measurement; we are not aware of any formal treatment of relative accuracy in the absence of true values.

Rules and master data have been used in repairing data (data consistency) [3,15] and specifying relative information completeness [12]. This work differs from the prior work in the following. (1) ARs are quite different from the dependencies used for specifying consistency and completeness. As a result, the termination problem for rules of [15], for instance, is PSPACE-complete, while our inference (chase) process always terminates. (2) Data repairing and information completeness consider problems different from those studied here. (3) We give an operational semantics for ARs in terms of chase [1]. In contrast, chase was not considered in the prior work except [3]. While [3] adapted chase for data repairing [3] based on matching dependencies, it studied neither how to deduce relative accuracy, nor the complexity of determining whether a chase process is Church-Rosser.

Also related is prior work on truth discovery from data sources [4, 8, 14, 19, 27–30]. Those approaches include (i) dependencies on sources to detect copy relationships and identify reliable sources [8]; (ii) employing lineage and probabilistic [27]; (iii) vote counting and probabilistic analysis based on the trustworthiness of data sources [4, 19, 28–30]. In contrast, we deduce relative accuracy following a logical approach based on ARs and master data, without assuming knowledge about data sources. Our method is *complementary to* the prior approaches for truth discovery, and can be combined with them by deducing trust in attributes with ARs in truth discovery (see Sections 3 and 7 for details).

Closer to this work is [14], on conflict resolution by reasoning about data consistency and currency. It used partial orders, currency constraints and constant CFDs [13]. This work differs from [14] in the following. (1) We study relative data accuracy rather than conflict resolution. This said, currency orders and constant CFDs can be expressed as ARs and hence, our techniques can also be used in data fusion [5]. (2) We use ARs for actions in a chase process, as opposed to static dependencies of [14]. (3) Our approach is quite different from [14]. We infer accuracy via chase, use available master data to improve accuracy, and provide algorithms to compute top-k target tuples. These were not studied in [14].

Our algorithms for computing top-k target tuples are related to top-k query answering, which aims to retrieve top-k tuples from query result, ranked by a monotone scoring function [11, 22]. One of our top-k algorithms extends algorithms for top-k rank join queries [21,26] by *embedding* score computation in top-k selection, rather than assuming that the scores are already given, and by additionally checking whether selected tuples observe ARs. We also provide a new algorithm that *does not* require the input to be ranked; it is instance optimal *w.r.t.* the number of visits to the data, and can be used to *compute rank joins on unranked lists*.

2. A MODEL FOR RELATIVE ACCURACY

We next present a model for determining relative accuracy. We first define ARs (Section 2.1), and then introduce a chase procedure for deducing relative accuracy (Section 2.2),

2.1 Rules for Specifying Relative Accuracy

Relative accuracy. Consider a relation schema $R = (A_1, \ldots, A_n)$, where the domain of attribute A_i is $\mathsf{dom}(A_i)$. We consider an *entity instance* I_e of R, which is a set of tuples pertaining to the *same* real-world entity e. Such an I_e is identified by entity resolution techniques [9, 24], and is typically *much smaller* than a database instance in practice.

The problem of relative accuracy is to determine, given an attribute A_i of R and tuples $t_1, t_2 \in I_e$, whether t_2 is *more accurate than* t_1 *in attribute* A_i, denoted by $t_1 \prec_{A_i} t_2$.

More specifically, for each attribute A_i of I_e, $\prec_{A_i}$ is a *strict partial order* defined on the A_i attribute values in I_e. That is, $\prec_{A_i}$ is a binary relation that is irreflexive and transitive, and thus asymmetric. Initially, $\prec_{A_i}$ is *empty* for all $i \in [1, n]$, and we want to populate $\prec_{A_i}$ by deducing relative accuracy with accuracy rules. We also use $t_1 \preceq_{A_i} t_2$ to denote either $t_1[A_i] = t_2[A_i]$ or $t_1 \prec_{A_i} t_2$. Note that $\preceq_{A_i}$ is a partial order, referred to as the *accuracy order on attribute* A_i.

Ultimately we want to find a tuple t_e for I_e, referred to as the *target tuple for e from* I_e such that for each attribute A_i of R and all tuples $t \in I_e$, $t \preceq_{A_i} t_e$. Intuitively, t_e is a *new tuple* composed of the most accurate value of attribute A_i for all $i \in [1, n]$. It is easy to verify that if t_e exists, then it is *unique*. Note that I_e may not have enough information for us to deduce a complete t_e. If so, $t_e[A_i] = \mathsf{null}$ for some A_i, and we refer to t_e as an *incomplete target tuple* of I_e.

Accuracy rules (ARs). There are two forms of ARs. The first one is defined on tuples $t_1, t_2 \in I_e$ to deduce whether $t_1 \preceq_{A_i} t_2$, *i.e.,* their relative accuracy in an attribute A_i:

$$\varphi = \forall t_1, t_2\ (R(t_1) \wedge R(t_2) \wedge \omega \rightarrow t_1\ \preceq_{A_i}\ t_2) \qquad (1)$$

where ω is a conjunction of predicates of the form: (a) $t_1[A_l]\ \mathsf{op}\ t_2[A_l]$, where op is one of the comparison operators $=, \neq, >, <, \leq, \geq$; or (b) $t_i[A_l]\ \mathsf{op}\ c$ for $i \in [1, 2]$, where c is a constant or $t_e[A_l]$; or (c) $t_1 \prec_{A_l} t_2$ or $t_1 \preceq_{A_l} t_2$. We refer to ω as $\mathsf{LHS}(\varphi)$ and $t_1 \preceq_{A_i} t_2$ as $\mathsf{RHS}(\varphi)$.

We denote by $(t_1, t_2) \models \omega$ if t_1 and t_2 satisfy the predicates in ω following the standard semantics of first-order logic. Intuitively, if $(t_1, t_2) \models \omega$, then $t_1 \prec_{A_i} t_2$ or $t_1 \preceq_{A_i} t_2$.

The second form of ARs is defined on (t_e, I_m), where t_e is the *target tuple template*, and I_m is an available master relation of schema R_m [25]. Note that R_m may not cover all the attributes of R. This form of ARs extends t_e by extracting accurate values from master relation I_m, as follows:

$$\varphi' = \forall t_m\ (R_m(t_m) \wedge \omega \rightarrow t_e[A_i] = t_m[B]) \qquad (2)$$

Here ω is a conjunction of predicates of the form $t_e[A_l] = c$ or $t_e[A_l] = t_m[B']$, where c is a constant and B' is an attribute of R_m. Intuitively, if t_e matches a master tuple t_m in I_m as specified by ω, then $t_e[A_i]$ is *instantiated* by taking the value of $t_m[B]$. We refer to ω as $\mathsf{LHS}(\varphi')$ and $t_e[A_i] = t_m[B]$ as $\mathsf{RHS}(\varphi)$. We write $(t_e, t_m) \models \omega$ if t_e and t_m satisfy ω.

Example 3: Recall the entity instance stat of Table 1, and master relation nba of Table 2. Then their ARs include φ_1–φ_6 given in Table 1. These ARs demonstrate how we can derive relative accuracy, in terms of (a) constants, built-in predicates and the semantics of the data such as φ_1, (b) data currency, *e.g.,* φ_2 and φ_3, (c) *co-existence* of attributes and known accuracy orders, such as φ_4 and φ_5, and (d) available master data such as φ_6. Additional ARs for stat include:

φ_7: $\forall t_1, t_2 \in \mathsf{stat}\ \big(t_1[A] = \mathsf{null} \wedge t_2[A] \neq \mathsf{null} \rightarrow t_1 \preceq_A t_2\big)$
φ_8: $\forall t_1, t_2 \in \mathsf{stat}\ \big(t_2[A] = t_e[A] \wedge t_e[A] \neq \mathsf{null} \rightarrow t_1 \preceq_A t_2\big)$
φ_9: $\forall t_1, t_2 \in \mathsf{stat}\ \big(t_1[A] = t_2[A] \rightarrow t_1 \preceq_A t_2\big)$
φ_{10}: $\forall t_1, t_2 \in \mathsf{stat}\ \big(t_1 \prec_{\mathsf{MN}} t_2 \rightarrow t_1 \preceq_{\mathsf{LN}} t_2\big)$
φ_{11}: $\forall t_1, t_2 \in \mathsf{stat}\ \big(t_1 \prec_{\mathsf{team}} t_2 \rightarrow t_1 \preceq_{\mathsf{arena}} t_2\big)$

Here φ_7–φ_9 are defined on all the attributes A of stat. Rule φ_7 says that the null value has the lowest accuracy; φ_8 asserts that if the target attribute $t_e[A]$ is defined, then it has the highest accuracy among all A-attribute values in I_e; and φ_9 says that for all t_1 and t_2, if $t_1[A] = t_2[A]$, then $t_1 \preceq_A t_2$. Rules φ_7–φ_9 are "axioms" that are included in any set of ARs. ARs φ_{10} and φ_{11} deduce accuracy from *correlated attributes* (*e.g.,* φ_{10}, if a tuple has a more accurate MN, then so does LN since the two attributes often come together).

Note that $t_1 \prec_A t_2$ iff $t_1 \preceq_A t_2$ and $t_1[A] \neq t_2[A]$. Hence when $\preceq_A$ is computed, we *can derive* $\prec_A$ *from* $\preceq_A$. □

Remark. Constant CFDs [13] developed for detecting data inconsistencies can be expressed as ARs. As an example, consider the CFD ψ given in Example 1: [team = "Chicago Bulls" → arena = "United Center"]. We can create a master relation of schema R_m with a tuple (team = "Chicago Bulls", arena = "United Center"), and express ψ as an AR

$\forall t_m \in \mathsf{R_m}\ (t_m[\mathsf{team}] = t_e[\mathsf{team}] \rightarrow t_e[\mathsf{arena}] = t_m[\mathsf{arena}])$,

which asserts that if the team of the target tuple t_e is Chicago Bulls, then its arena must be United Center. As we only need to assure the consistency of the target tuple t_e, general CFDs defined on two tuples are not needed here.

2.2 Inferring Relative Accuracy

We next present an inference system for relative accuracy, in terms of a chase-like procedure with ARs. The chase process gives an operational semantics for ARs.

We start with some notations. Consider an entity instance I_e, a master relation I_m, and a set Σ of ARs defined on I_e and I_m. (1) We use D to denote $(I_e, \preceq_{A_1}, \ldots, \preceq_{A_n})$, *i.e., I_e* equipped with partial orders $\preceq_{A_i}$; we use $\preceq_{A_i}^{D}$ to denote the partial order $\preceq_{A_i}$ in D [3]. (2) We call $(D_0, t_e^{D_0})$ the *initial instance* of e, where $\preceq_{A_i}^{D_0}$ is empty, and $t_e^{D_0}$ is the target template with $t_e^{D_0}[A_i] = \mathsf{null}$ for all $i \in [1, n]$. (3) We refer to $S = (D_0, \Sigma, I_m, t_e^{D_0})$ as a *specification of entity e*. (4) We call (D, t_e^{D}) an *accuracy instance* of S, where t_e^{D} is *the target tuple template associated with D*, which is instantiated in the chase process, and may have null in some of its attributes.

In a nutshell, the chase starts with the initial instance $(D_0, t_e^{D_0})$. It deduces relative accuracy by *populating partial orders* and *instantiating the target tuple template*, yielding a sequence $(D_0, t_e^{D_0})$, $(D_1, t_e^{D_1})$, …, $(D_m, t_e^{D_m})$ of accuracy

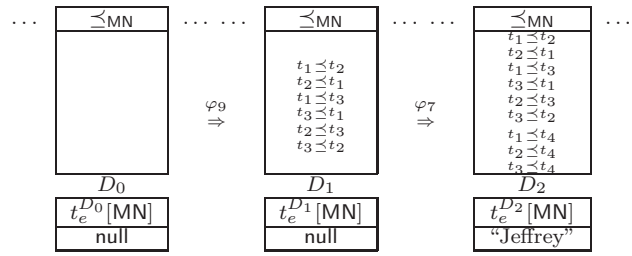


Figure 2: Single chase steps

instances. Each chase step applies an AR $\varphi \in \Sigma$ and I_m to an accuracy instance $(D_j, t_e^{D_j})$, and generates another instance $(D_{j+1}, t_e^{D_{j+1}})$. In other words, $(D_{j+1}, t_e^{D_{j+1}})$ is an updated version of $(D_j, t_e^{D_j})$, such that for some attribute A_i, either the partial order $\preceq_{A_i}^{D_{j+1}}$ extends $\preceq_{A_i}^{D_j}$ with a new pair, or $t_e^{D_j}[A_i] = \mathsf{null}$ is instantiated by letting $t_e^{D_{j+1}}[A_i]$ take a value from a master tuple in I_m or a value that is already determined most accurate for A_i. The process proceeds until no changes can be made to partial orders or the target tuple template. More specifically, these are stated as follows.

(1) A single chase step. We say that $(D_{j+1}, t_e^{D_{j+1}})$ is *an immediate result* of enforcing an AR $\varphi \in \Sigma$ on $(D_j, t_e^{D_j})$ with I_m, denoted by $(D_j, t_e^{D_j}) \mapsto_\varphi (D_{j+1}, t_e^{D_{j+1}})$, if $(D_j, t_e^{D_j}) \neq (D_{j+1}, t_e^{D_{j+1}})$ and one of the following conditions holds:

(a) When $\varphi = \forall t_1, t_2 (R(t_1) \wedge R(t_2) \wedge \omega \rightarrow t_1 \preceq_{A_i} t_2)$. Then there exist tuples $t_1, t_2 \in I_e$ such that

- $(t_1, t_2) \models \omega$;
- $D_{j+1} = (I_e, \preceq_{A_1}^{D_j}, \cdots, \preceq_{A_i}^{D_j} \cup \{(t_1, t_2)\}, \cdots, \preceq_{A_n}^{D_j})$; and
- $t_e^{D_{j+1}} = (t_e^{D_j}[A_1], \cdots, \lambda(t_e^{D_j}[A_i], \preceq_{A_i}^{D_{j+1}}) \cdots, t_e^{D_j}[A_n])$.

Here $\lambda(t_e^{D_j}[A_i], \preceq_{A_i}^{D_j}) = t[A_i]$ if there exists $t \in I_e$ such that for all $t' \in I_e$, $t' \preceq_{A_i}^{D_{j+1}} t$; and it is $t_e^{D_j}[A_i]$ otherwise.

Intuitively, $\preceq_{A_i}^{D_{j+1}}$ extends $\preceq_{A_i}^{D_j}$ by including $t_1 \preceq_{A_i} t_2$, and $t_e^{D_{j+1}}[A_i]$ takes the A_i value with the highest accuracy *w.r.t.* $\preceq_{A_i}^{D_{j+1}}$ if it exists. Note that D_{j+1} and D_j agree on every attribute and partial order other than $t_e^{D_j}[A_i]$ and $\preceq_{A_i}^{D_j}$.

(b) When $\varphi = \forall t_m\ (R_m(t_m) \wedge \omega \rightarrow t_e[A_i] = t_m[A_i])$. Then there exist tuples $t \in I_e$ and $t_m \in I_m$ such that

- $(t_e, t_m) \models \omega$, $D_{j+1} = D_j$, and
- $t_e^{D_{j+1}} = (t_e^{D_j}[A_1], \cdots, t_m[A_i], \cdots, t_e^{D_j}[A_n])$.

Here $t_e^{D_{j+1}}$ differs from $t_e^{D_j}[A_i]$ only in attribute A_i by taking master data $t_m[A_i]$, while D_{j+1} remains unchanged from D_j.

We say that $(D_j, t_e^{D_j}) \mapsto_\varphi (D_{j+1}, t_e^{D_{j+1}})$ is *valid* if (a) there exist no t_1 and t_2 such that both $t_1 \prec_{A_i}^{D_{j+1}} t_2$ and $t_2 \prec_{A_i}^{D_{j+1}} t_1$ (*i.e.,* $t_1 \preceq_{A_i}^{D_{j+1}} t_2$, $t_2 \preceq_{A_i}^{D_{j+1}} t_1$ but $t_1[A_i] \neq t_2[A_i]$), and (b) $t_e^{D_j}[A_i]$ is not changed if $t_e^{D_j}[A_i] \neq \mathsf{null}$. In the chase process we consider valid chase steps only.

Observe the following: (a) the entity instance I_e and the master data I_m remain unchanged when ARs are enforced; (b) $\preceq_{A_i}^{D_j}$ and $\preceq_{A_i}^{D_{i+1}}$ are partial orders for all attributes A_i, such that for all $t_1, t_2 \in I_e$, if $t_1 \preceq_{A_i}^{D_j} t_2$ and $t_2 \preceq_{A_i}^{D_j} t_1$ then $t_1[A_i] = t_2[A_i]$; and (c) if $t_e^{D_j}[A_i] \neq \mathsf{null}$, then $t_e^{D_{j+1}}[A_i] = t_e^{D_j}[A_i]$, *i.e.,* all non-null values of $t_e^{D_j}$ *remain unchanged.*

Example 4: Consider I_e = stat (Table 1), I_m = nba (Table 2), and Σ consisting of the ARs given in Example 3. Let D_0 be I_e with empty partial orders, and $t_e^{D_0}$ be the initial

target template with $t_e^{D_0}[A] = \mathsf{null}$ for all attributes A. After enforcing φ_9 on $(D_0, t_e^{D_0})$, it yields $(D_1, t_e^{D_1})$ as the first step in Fig. 2, in which $\preceq_{\mathsf{MN}}$ is extended on (t_1, t_2), (t_2, t_3) and (t_3, t_1). Similarly, after enforcing φ_7 on $(D_1, t_e^{D_1})$, it yields $(D_2, t_e^{D_2})$ as the second step in Fig. 2, which extends $\preceq_{\mathsf{MN}}$ on (t_1, t_4) and instantiates $t_e^{D_2}[\mathsf{MN}] =$ "Jeffrey". □

(2) Chase. A *chasing sequence* of D_0 by Σ and I_m is a sequence of accuracy instances $(D_0, t_e^{D_0})$, $(D_1, t_e^{D_1})$, ..., $(D_l, t_e^{D_l})$, ..., where for each $j \geq 1$, there exists some AR $\varphi \in \Sigma$ such that $(D_j, t_e^{D_j}) \mapsto_\varphi (D_{j+1}, t_e^{D_{j+1}})$ is valid.

A chasing sequence $(D_0, t_e^{D_0})$, ..., $(D_k, t_e^{D_k})$ is said to be *terminal* if it is finite and moreover, no more valid step can be enforced on $(D_k, t_e^{D_k})$. We refer to $t_e^{D_k}$ as a *deduced target tuple* of specification S, and $(D_k, t_e^{D_k})$ as the *terminal instance* of the chasing sequence.

Intuitively, the chase repeatedly applies ARs to deduce relative accuracy and instantiate the target tuple template, until it reaches an instance that cannot be further changed.

Example 5: Consider $D_0, t_e^{D_0}, \Sigma$ and I_m given in Example 4. By enforcing ARs φ_9, φ_7, φ_5, φ_{10}, φ_6, φ_1, φ_4, φ_2, φ_3 and φ_{11} on $(D_0, t_e^{D_0})$ in this order with I_m, one can get a deduced target tuple t_e[FN, MN, LN, rnds, totalPts, J#, league, team, arena] = (Michael, Jeffrey, Jordan, 27, 772, 23, NBA, Chicago Bulls, United Center). Note that t_e is a complete target tuple from stat, which draws values from *different tuples*, *e.g.,* t_2 and t_4 of stat and s_1 of nba. □

3. FUNDAMENTAL PROBLEMS

Given a specification $S = (D_0, \Sigma, I_m, t_e^{D_0})$ of an entity e, we want to know whether chasing on D_0 by Σ and I_m terminates? Whether will all chasing sequences of D_0 lead to the same deduced target tuple t_e? When the target tuple t_e is incomplete, can we make it complete while observing the ARs in Σ? Can we find top-k candidate targets for users to chose? This section studies these issues (due to the lack of space we defer all the proofs to the full version). As will be seen in Section 4, our framework for deducing relative accuracy and target tuples are based on these results.

(1) Termination of chase. Is every chasing sequence of D_0 by Σ and I_m an initial subsequence of a terminal chasing sequence? The answer to this question is affirmative.

Proposition 1: *Every chasing sequence of D_0 by Σ and I_m is finite and leads to a terminal instance in $O(|I_e|^2)$ steps, where $|I_e|$ is the size of the entity instance I_e in D_0.* □

(2) The Church-Rosser property. Another question asks whether different terminal chasing sequences of D_0 by Σ and I_m lead to *the same unique terminal instance*, no matter what rules in Σ are used and in what order they are applied. This is known as the *Church-Rosser property* (see, *e.g.,* [1]). If a specification S has the Church-Rosser property, we say that S *is Church-Rosser*. Obviously if S is Church-Rosser, then the uniquely deduced target tuple is *deterministic*, yielding a unique target that can be "trusted".

Unfortunately, not all specifications are Church-Rosser.

Example 6: Consider the specification S described in Example 5. One can verify that S is Church-Rosser. However, let us extend S to S' by adding an extra rule φ_{12}: $\forall t_1, t_2 \in \mathsf{stat}\ \big(t_1[\mathsf{league}] = \text{NBA} \wedge t_2[\mathsf{league}] = \text{SL} \rightarrow t_1 \preceq_{\mathsf{league}} t_2\big)$. Then S' is not Church-Rosser. Indeed, there are two chasing sequences that deduce different target tuples: one is the sequence given in Example 5 with $t_e[\mathsf{league}] = \text{NBA}$, and the other is by enforcing ARs φ_7, φ_5, φ_{10} and φ_{12} in this order, yielding a target tuple t'_e with $t'_e[\mathsf{league}] = \text{SL}$. □

This tells us that if S is not Church-Rosser, it may lead to multiple conflicting targets (*e.g.,* t_e and t'_e on league), which cannot be accurate at the same time. Thus specifications that are not Church-Rosser should be *identified and revised*.

To do this, we provide *a necessary and sufficient condition* for deciding whether a specification S is Church-Rosser. We say that a terminal chasing sequence $(D_0, t_e^{D_0})$, ..., $(D_k, t_e^{D_k})$ is *stable* if for all *invalid* chase steps that enforce an AR φ on $(D_k, t_e^{D_k})$, φ cannot be enforced on $(D_j, t_e^{D_j})$ as a valid step for all $j \in [0, k-1]$. That is, suppose that $(D_k, t_e^{D_k})$ can be further changed by φ by letting $t_e^{D_k}[A_i]$ change from a non-null value to another, or by allowing both $t_1 \preceq_{A_i}^{D_k} t_2$ and $t_2 \preceq_{A_i}^{D_k} t_1$ while $t_1[A_i] \neq t_2[A_i]$. Then the change cannot be inflicted to any $(D_j, t_e^{D_j})$ as a valid move. Intuitively, if φ could be enforced as a valid step, it would lead to a terminal sequence *different* from $(D_k, t_e^{D_k})$.

A stable chasing sequence prevents any conflicts in the chase such as those in Example 6, and allows us to efficiently determine whether S is Church-Rosser. In light of this, in the sequel we focus on Church-Rosser specifications only.

Theorem 2: *Given a specification $S = (D_0, \Sigma, I_m, t_e^{D_0})$, (a) S is Church-Rosser if and only if there exists a terminal chasing sequence of S that is stable; and (b) it is in $O((|I_e|^2 + |I_m|)|\Sigma|)$ time to decide whether S is Church-Rosser.* □

(3) Deducing candidate targets. When S is Church-Rosser, its deduced target tuple t_e may still be *incomplete*, *i.e.,* some attributes remain null. For example, if we drop AR φ_{11} of Example 3 from the specification of Example 5, the reduced specification is still Church-Rosser, but its deduced target is incomplete since the most accurate value of arena can no longer be determined, as indicated in Example 2.

This gives rise to the following question: can we find candidate targets and suggest them for the users to consider? More specifically, a complete tuple t'_e is called a *candidate target* of a specification $S = (D_0, \Sigma, I_m, t_e^{D_0})$ if

- for each attribute A_i, $t'_e[A_i] = t_e[A_i]$ if $t_e[A_i] \neq \mathsf{null}$, and $t'_e[A_i]$ is a value in $\mathsf{dom}(A_i)$ otherwise, where t_e is the unique deduced target tuple of S;
- $S' = (D_0, \Sigma, I_m, t'_e)$ is Church-Rosser and moreover, t'_e is the deduced target tuple of S'.

That is, a candidate target t'_e keeps the non-null values of t_e unchanged but instantiates those null attributes of t_e. Moreover, when we treat t'_e as the initial target template, the chase verifies that t'_e "satisfies" the constraints imposed by the ARs of Σ, and is deduced as the target tuple of S'.

The *candidate target problem* is to determine, given a specification S of an entity that is Church-Rosser, whether there exists a candidate target t'_e of S. It is, however, nontrivial.

Theorem 3: *The candidate target problem is NP-complete. It remains NP-hard for specifications $S = (D_0, \Sigma, I_m, t_e^{D_0})$ in which Σ consists of ARs of form (1) only, and when candidate targets t_e of S take values from I_e and I_m only.* □

The number of candidates t'_e for a Church-Rosser S could be quite large, exponential or even infinite.

Example 7: Consider $R = (A_1, \ldots, A_n)$, an entity instance I_e of R with tuples $t_1 = (0, \ldots, 0)$ and $t_2 = (1, \ldots, 1)$, and empty Σ and I_m. Then there are 2^n candidate targets with

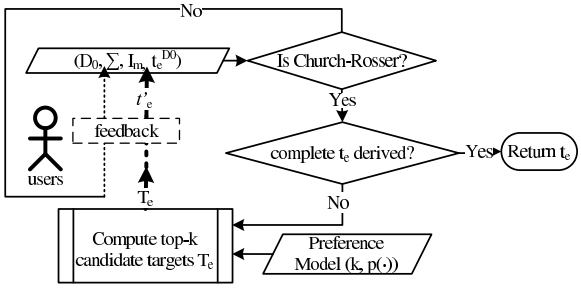


Figure 3: Framework overview

values from $\{0,1\}$, *i.e.,* each tuple $t \in \{0,1\}^n$ is a candidate target. Worse still, if some A_i of R has an infinite domain, there are possibly *infinitely many* candidate targets. □

(4) Finding top-k candidate targets. It is infeasible to enumerate all candidate targets. This suggests that we find top-k candidate targets for S based on a preference model.

We specify the preference model as a pair $(k, p(\cdot))$, where k is a natural number, and $p(\cdot)$ is a monotone scoring function such that given a set T_e of candidate targets, $p(T_e)$ is a real number. To simplify the discussion we assume that a real number $w_{A_i}(v)$ is associated with each value v in domain $\mathsf{dom}\ (A_i)$ (if $\mathsf{dom}\ (A_i)$ is infinite, $w_{A_i}(v)$ is the same for all v outside of I_e and I_m), referred to as the *score of* v. The score could be placed by the users as the confidence in v [12], found as probabilities by truth discovery algorithms [4, 19, 28–30] (see Section 7), or automatically derived by counting the occurrences of v in the A_i column and from co-existence of attributes and available scores. We define the *score*

$$p(T_e) = \sum_{t'_e \in T_e} \sum_{A_i \in \mathsf{attr}(R)} w_{A_i}(t'_e[A_i]).$$

Such preference is often too "soft" to be modeled as ARs or partial orders, and candidates derived from it may not be as "deterministic" ("certain") as deduced targets by the chase. Nonetheless, users often find such candidates helpful, as commonly practiced in data repairing heuristics [12].

For a Church-Rosser S, a preference model $(k, p(\cdot))$ and a number C, the *top-k candidate problem* is to decide whether there exists a set T_e of k candidate targets with $p(T_e) \geq C$.

Theorem 4: *The top-k candidate problem is* NP*-complete, and* NP*-hard under the same restriction of Theorem 3.* □

4. A FRAMEWORK

We now present a framework for deducing complete target tuples for entities. As depicted in Fig. 3, given a specification $S = (D_0, \Sigma, I_m, t_e^{D_0})$ of an entity e, it populates partial orders for relative accuracy and instantiates the target tuple template, based on the chase given in Section 2. It automatically deduces as many accurate values for e as possible, and interacts with the users to revise candidate targets, until a complete target tuple is found. It works as follows.

(1) *Church-Rosser checking.* It first inspects whether S is Church-Rosser via *automated reasoning.* The Church-Rosser property warrants a unique target tuple, in which the accurate values can be trusted (Section 3). If S is not Church-Rosser, the users are invited to revise S (see step (4) below), by following the "No" branch. The revised S is then checked.

(2) *Computing target tuple* t_e*.* When S is confirmed Church-Rosser, the framework computes the unique deduced target tuple t_e by means of the chase. It returns t_e if it is complete. Otherwise it computes a top-k set of candidate targets.

(3) *Computing top-k candidate targets.* As remarked in Section 3, t_e may be incomplete, and it is hard to identify (all) candidate targets (Theorem 3 and Example 7). To this end, the framework computes a top-k set T_e of candidate targets. It comes up with a preference model $(k, p(\cdot))$ following the practice of data repairing heuristics, which the users may opt to adjust. Based on $(k, p(\cdot))$, it computes T_e with k tuples such that (a) for each $t'_e \in T_e$, t_e is a candidate target of S, and (b) for all sets T'_e with k candidate targets, $p(T_e) \geq p(T'_e)$, *i.e.,* tuples in T_e have the highest scores. When there exist at least k candidate targets of S, T_e consists of k distinct tuples; otherwise T_e includes all candidate targets of S. The set T_e is then suggested to the users.

(4) *User feedback.* The users are invited to inspect T_e. They may opt to choose some $t'_e \in T_e$ as the target tuple (recall that for each candidate target t'_e, $t'_e[A] = t_e[A]$ if $t_e[A] \neq \mathsf{null}$); or revise S by instantiating $t_e[B]$ with either the value of some $t'_e[B]$ or a value $v \in \mathsf{dom}(B)$, for some $t_e[B] = \mathsf{null}$. The users are also allowed to revise S by editing ARs in Σ and tuples in I_e (D_0). The revised S with the designated initial values is then checked by step (1).

The process proceeds until a complete t_e is found.

In the rest of the paper we will provide algorithms underlying the framework: an algorithm for checking the Church-Rosser property of S and deducing t_e in Section 5, and algorithms for computing top-k candidate targets in Section 6.

Remark. (1) To find ARs as input of the framework, we need algorithms for discovering ARs from (*possibly dirty*) data. ARs of type (2) can be discovered along the same lines as matching dependencies (see, *e.g.,* [12] for a survey). ARs of type (1) could be found by mining first-order logic rules (*e.g.,* [17]). Given a relation r of schema R, one may also group pairs of its tuples (t_i, t_j) into classes based on their attribute values $(t_i[A], t_j[A])$ $(t_i, t_j \in r,\ A \in \mathsf{attr}(R))$ to denote accuracy orders, and discover ARs by analyzing the containment of those classes via a level-wise approach (*e.g.,* [7]). We defer a full treatment of AR discovery to future work.

(2) The framework can handle *possibly dirty* entity instances. Indeed, constant CFDs [13] for detecting data inconsistencies can be expressed as ARs (Section 2). Thus in the same framework the consistency of target tuples can be assured. The framework can also incorporate data repairing algorithms, which have been well studied (see, *e.g.,* [12]).

5. CHECK CHURCH-ROSSER PROPERTY

We next present an algorithm that, given a specification $S = (D_0, \Sigma, I_m, t_e^{D_0})$, checks whether S is Church-Rosser. If so, it computes the unique terminal instance (D, t_e), and returns *nil* otherwise. The algorithm is in $O((|I_e|^2 + |I_m|)|\Sigma|)$ time, and thus gives a constructive proof for Theorem 2(2).

The algorithm is denoted by IsCR and shown in Fig. 4. Following Theorem 2(1), IsCR checks whether S has a stable terminal chasing sequence, by simulating the chase. At each step of the chase, it collects all valid steps in a set $\mathcal{Q}$, and when the chase process proceeds, it checks whether any valid step in $\mathcal{Q}$ becomes invalid. If so, it concludes that S is not Church-Rosser. Indeed, an invalid step for an instance *remains invalid* in the rest of the chasing sequence. Hence if a valid step becomes invalid later, it will not lead to any stable terminal chasing sequence. If IsCR inspects all valid steps and if none of them becomes invalid, it actually iden-

tifies a stable terminal sequence, and thus concludes that S is Church-Rosser. In the process (D, t_e) is also constructed.

Algorithm IsCR makes use of (1) a procedure, denoted by Instantiation, as a preprocessing step to identify all single chase steps, and (2) an indexing structure $\mathcal{H}$ for efficiently locating applicable chase steps, described as follows.

Computing single chase steps. Procedure Instantiation pre-computes possible single chase steps, collected in a set Γ, by partially evaluating each AR $\varphi \in \Sigma$ on tuples in the entity instance I_e of D_0 and master relation I_m, as follows.

(1) When φ is of form (1) $\forall t_1, t_2(\omega \rightarrow t_1 \preceq_{A_k} t_2)$, for each pair (t_i, t_j) of tuples in I_e, it computes $\phi = (\omega' \rightarrow t_i \preceq_{A_k} t_j)$, where ω' is obtained by evaluating $\omega(t_i, t_j)$, which substitutes (t_i, t_j) for (t_1, t_2) in ω. More specifically, for each predicate in ω, (a) if it is of the form $t_1[A_l]$ op $t_2[A_l]$ or $t_s[A_s]$ op c ($s \in [1, 2]$), where op is one of $=, \neq, >, <, \leq, \geq$, then the predicate on (t_i, t_j) evaluates to true or false. If it is true, the predicate is not included in ω'. If it is false, ω' also becomes false. (b) If it is $t_1 \prec_{A_l} t_2$ (resp. $t_1 \preceq_{A_l} t_2$), the predicate is replaced by $t_i \prec_{A_l} t_j$ (resp. $t_i \preceq_{A_l} t_j$) in ω'. We include ϕ in Γ if ω' is not false. Intuitively, ϕ indicates a single chase step: if ω' is satisfied, then $t_i \preceq_{A_k} t_j$ could be deduced.

(2) When $\varphi = \forall t_m\ (R_m(t_m) \wedge \omega \rightarrow t_e[A_i] = t_m[B])$, *i.e.,* of form (2), for each $t' \in I_m$, it computes $\phi = (\omega' \rightarrow t_e[A_i] = c)$, where t_e is the target template, c is the constant $t'[B]$, and ω' is obtained from $\omega(t_m)$ by substituting constant $t'[B']$ for each $t_m[B']$ in ω. We include ϕ in Γ, which indicates that if ω' is satisfied, then $t_e[A_i]$ can be instantiated with c.

Note that no ϕ in Γ carries the $\forall$ quantifier. Moreover, each chase step can be carried out by enforcing some ϕ in Γ rather than ARs in Σ. We use $\mathsf{LHS}(\phi)$ to denote ω'.

Example 8: The following single chase steps can be derived: (a) *true* $\rightarrow 16 \prec_{\mathsf{rnds}} 27$ from t_1, t_2 of Table 1 and φ_1 (form (1)) of Table 3; (b) $16 \prec_{\mathsf{rnds}} 27 \rightarrow 45 \prec_{\mathsf{J\#}} 23$ from t_1, t_2 and φ_2 (form (1)); and (c) $t_e[\mathsf{FN}, \mathsf{LN}] =$ (Michael, Jordan) $\rightarrow$ $t_e[\mathsf{league}, \mathsf{team}] =$ (NBA, Chicago Bulls) from master tuple s_1 of Table 2 and φ_6 (form (2)) of Table 3. □

Building Indices. Algorithm IsCR uses an indexing structure $\mathcal{H}$ to speed up the process of finding next applicable chase steps. The structure $\mathcal{H}$ is defined as follows.

(1) For each $\phi \in \Gamma$, $\mathcal{H}$ contains a counter n_ϕ to keep track of the number of predicates in $\mathsf{LHS}(\phi)$ that are not yet satisfied.

(2) For each predicate δ of the form of either $t_i \preceq_{A_k} t_j$ or $t_e[A_k] = c$, $\mathcal{H}$ maintains a set $\Phi_\delta = \{\phi \mid \phi \in \Gamma \wedge \delta \in \mathsf{LHS}(\phi)\}$, *i.e.,* the set of ϕ's in Γ that contain δ in $\mathsf{LHS}(\phi)$.

(3) A set $\mathcal{Q}$ is maintained by $\mathcal{H}$, which consists of all applicable single chase steps that were once valid. Initially, $\mathcal{Q} = \{\phi \mid \phi \in \Gamma \wedge n_\phi = 0 \wedge \phi$ was a valid step for $(D_0, t_e^{D_0})\}$.

Algorithm. We now present the main driver of IsCR. Given S, it first identifies all possible single steps and builds the index $\mathcal{H}$, by invoking procedures Instantiation and InitIndex (not shown), respectively (lines 1-2). It then initializes the accuracy instance template (D, t_e) with $(D_0, t_e^{D_0})$ (line 3).

After these, IsCR simulates the chasing of S (lines 3-13). When the set $\mathcal{Q}$ in $\mathcal{H}$ of valid steps is nonempty, it picks an applicable step ϕ from $\mathcal{Q}$ using a procedure NextStep (line 5, not shown), which removes ϕ from $\mathcal{Q}$. It then enforces ϕ as follows. If ϕ is derived from an AR of form (1) (lines 6-8), IsCR adds the derived partial order to D (line 7), and deduces $t_e[A_k]$ whenever possible (line 8). If ϕ is derived from

Input: A specification $S = (D_0, \Sigma, I_m, t_e^{D_0})$.
Output: The unique terminal instance (D, t_e) if S is Church-Rosser, and *nil* otherwise.

1. $\Gamma :=$ Instantiation(D_0, Σ, I_m);
2. $\mathcal{H} :=$ InitIndex$(\Gamma, D_0, t_e^{D_0})$; /*$\mathcal{Q}$ in $\mathcal{H}$ maintains single steps*/
3. $D := D_0$; $t_e := t_e^{D_0}$;
4. **repeat**
5. $\quad\phi :=$ NextStep$(\mathcal{H})$;
6. $\quad$**if** $\phi = (\omega \rightarrow t_i \preceq_{A_k} t_j)$ (*i.e.,* form (1)) **then**
7. $\quad\quad D :=$ the transitive closure of $D \cup \{t_i \preceq_{A_k} t_j\}$;
8. $\quad\quad$Update $t_e[A_k]$;
9. $\quad$**if** $\phi = (\omega \rightarrow t_e[A_i] = c)$ (*i.e.,* form (2)) **then**
10. $\quad\quad t_e[A_i] := c$;
11. $\quad$**if not** IsValid(ϕ, D, t_e) **then return** *nil*;
12. $\quad$Update $\mathcal{H}$;
13. **until** $\mathcal{Q}$ in $\mathcal{H}$ becomes empty $\emptyset$;
14. **return** (D, t_e)

Figure 4: Algorithm IsCR

an AR of form (2) (lines 9-10), IsCR sets $t_e[A_k] := c$ (line 10). If ϕ is invalid for (D, t_e), we can conclude that S is not Church-Rosser since there will be no stable terminal chasing sequences, as argued above, and IsCR returns *nil* (line 11). Otherwise IsCR updates $\mathcal{H}$ to reflect the changes to D and t_e (line 12): for each $t_i \preceq_{A_k} t_j$ derived, $n_{\phi'}$ is decreased by 1 for each $\phi' \in \Phi_{t_i \preceq_{A_k} t_j}$; and for each $t_e[A_k] = c$ derived, it decreases $n_{\phi'}$ by 1 for each $\phi' \in \Phi_{t_e[A_k]=c}$. For any ϕ' with $n_{\phi'} = 0$, ϕ' is added to $\mathcal{Q}$, *i.e.,* it now becomes a valid chase step, to be considered later. The process proceeds until no more steps in $\mathcal{Q}$ need to checked (line 13), and it returns (D, t_e) as the terminal instance (line 14).

Correctness & complexity. The correctness of IsCR follows from Theorem 2(1) and the argument above, since it checks all possible chase steps that are valid at some point of chasing. For the complexity, observe the following. (1) Instantiation is in $O(|\Sigma|(|I_e|^2 + |I_m|))$ time, which is also the bound on $|\Gamma|$. After Instantiation, IsCR no longer needs to visit I_e. (2) With the indices, NextStep takes O(1) time, and each ϕ is checked only once. (3) Each step derives new partial orders $\preceq_{A_k}$ and/or instantiates $t_e[A_k]$. Thus, the total number of steps processed (lines 5-12) is bounded by $O(|I_e|^2)$. Therefore, IsCR is in $O(|\Sigma|(|I_e|^2 + |I_m|))$ time. As remarked earlier, I_e is *much smaller* than a database instance. As will be seen in Section 7, IsCR takes about 10ms.

6. FIND TOP-k CANDIDATE TARGETS

We next provide algorithms that, given a Church-Rosser specification $S = (D_0, \Sigma, I_m, t_e^{D_0})$ and a preference model $(k, p(\cdot))$, compute a set T_e of top-k candidate targets. Here T_e consists of k distinct candidate targets of S with the maximum score $p(T_e)$ if there exist at least k such tuples of S; and otherwise T_e includes all candidate targets of S.

Theorem 4 tells us that the top-k candidate problem is NP-complete. Worse still, there exists no PTIME algorithm for it with a bounded approximation-ratio unless P = NP. Recall that NPO is the *class* of all NP optimization problems. An NPO-complete problem is NP-hard to optimize, and is among the hardest optimization problems.

Theorem 5: *The top-k candidate targets problem (optimization version) is* NPO*-complete.* □

Despite the hardness, we provide three algorithms to find top-k candidate targets, all with the *early termination* property, *i.e.,* they stop as soon as top-k candidate targets are found. The first two are exact algorithms. (1) The first one,

RankJoinCT, extends prior algorithms [21, 26] for computing top-k joins of ranked lists (Section 6.1). (2) The second one, TopKCT, is developed for a more general setting when the ranked lists are *not* given (Section 6.2). We show that TopKCT incurs *lower cost* than RankJoinCT, and is *instance optimal w.r.t.* the number of visits to the data needed. (3) The third one, TopKCT$_h$, is a PTIME heuristic version of TopKCT (Section 6.3). We also identify special cases when the top-k candidate targets problem is in PTIME.

6.1 RankJoinCT: An Algorithm based on Rank Join

Given a set of ranked lists and a monotone scoring function, *the top-k rank join problem* is to compute the top k join results of the lists with the highest scores. Our problem can be modeled as an extension of the top-k rank join problem as follows. Consider a Church-Rosser specification S, of which t_e is the unique deduced target tuple. Let Z be the set of attributes A such that $t_e[A_i] = \mathsf{null}$. Assume *w.l.o.g.* that Z consists of m attributes $A_1, \cdots, A_m$. Then a set T_e of top-k candidate targets is a set of top-k join results of values in each domain of Z such that it satisfies an *additional condition*: for each $t \in T_e$, the revised specification $S' = (D_0, \Sigma, I_m, t'_e)$ must also be Church-Rosser, where $t'_e[Z] = t[Z]$ and $t'_e[B] = t_e[B]$ for all $B \in R \setminus Z$.

In light of this, we develop algorithm RankJoinCT by extending top-k rank join algorithms [21, 26]. The algorithm assumes that the domain values of Z attributes are *ranked* based on their scores (see $w_{A_i}(\cdot)$ in Section 3). It takes as input S, $(k, p(\cdot))$, t_e and moreover, m lists $L_1, \cdots, L_m$, such that L_i is the ranked list of values in the *active domain* of A_i for all $i \in [1, m]$. It returns a top-k *list* T_e of candidate targets as required. Note that L_i is *finite*: the active domain of A_i is $\mathsf{dom}(A_i)$ if it is finite; otherwise it includes all A_i values from I_e or I_m and at most one more distinct value from $\mathsf{dom}(A_i)$, which suffices to denote values outside of I_e or I_m and is referred to as a *default value* (see Section 3). We omit the details of RankJoinCT for the lack of space, but give an example below to illustrate how it works.

Example 9: Consider the specification S of Example 5 and a preference model $(k = 2, p(\cdot))$, where $p(\cdot)$ counts value occurrences. Suppose that we drop team from φ_6 of Table 3. Then the deduced target t_e is incomplete since t_e[team] and t_e[arena] become null. To find top 2 candidate targets for S, RankJoinCT takes as input the modified S, $(2, p(\cdot))$ and two ranked list L_{team} and L_{arena} with $\perp_{\mathsf{team}}$ and $\perp_{\mathsf{arena}}$ as default values for team and arena, respectively. It maintains upper bounds u_{team} and u_{arena}, asserting that for all tuples with values in L_{team} (resp. L_{arena}), their scores are no higher than u_{team} (resp. u_{arena}). RankJoinCT iteratively retrieves candidate targets from the two lists. Initially, it picks top values for team and arena, and forms a candidate target t_1 with t_1[team, arena] = (Chicago Bulls, United Center) ($p(t_1) = 4$). It then updates both u_{team} and u_{area} to 3. After that, it picks the next top unseen value v from L_{arena} and chooses a candidate target from the join results of $\{v\}$ with other fixed values in t_1, which is t_2 with t_2[team, arena] = (Chicago Bulls, Chicago Stadium (or Regions Park)) ($p(t_2)$ = 3). Now $p(t_1)$ and $p(t_2)$ are no less than u_{FN} and u_{arena}.

Further, for each $t \in \{t_1, t_2\}$, RankJoinCT has to check the condition additional to [21,26], *i.e.,* whether t is a candidate target. This is done by procedure check, which is essentially IsCR of Fig. 4 by taking t as the initial target. As t_1 and t_2 pass check, they are returned as top-2 candidates. □

Input: S, $(k, p(\cdot))$ and t_e as for RankJoinCT, and heaps $H_1, \ldots, H_m$.
Output: A list T_e of top-k candidate targets of S.

1. $T_e := nil$;
2. **for each** $i \in [1, m]$ **do** $B_i := [H_i.\mathsf{pop}()]$;
3. $o.t[Z] := (B_1[0], \cdots, B_m[0])$; $o.t[R \setminus Z] := t_e[R \setminus Z]$;
4. **for each** $i \in [1, m]$ **do** $o.p_i := 0$;
5. $o.w := p(\{o.t\})$;
6. $T := \{o.t\}$; $Q := \mathsf{BrodalQueue}(\{o\})$;
7. **while** $\|T_e\| < k$ **and** $\|Q\| > 0$ **do**
8. $o := Q.\mathsf{pop}()$;
9. **if** $\mathsf{check}(o.t, S)$ **then** $T_e.\mathsf{append}(o.t)$; /* add $o.t$ to T_e*/
10. **for each** $i \in [1, m]$ **do**
11. **if** $\mathrm{len}(B_i) \leq o.p_i + 1$ **then** $B_i.\mathsf{append}(H_i.\mathsf{pop}())$;
12. $o' := o$;
13. $o'.p_i := o'.p_i + 1$; $o'.t[A_i] := B_i[o'.p_i]$;
14. $o'.w := o.w - w_{A_i}(o.t[A_i]) + w_{A_i}(o'.t[A_i])$;
15. **if** $o'.t \notin T$ **then** $Q.\mathsf{push}(o')$; $T := T \cup \{o'.t\}$;
16. **return** T_e;

Figure 5: Algorithm TopKCT

Following [21], one can readily verify:

Proposition 6: RankJoinCT *finds top-k candidate targets with the early termination property,* i.e., *it does not need to check all tuples in the product of ranked lists.* □

However, RankJoinCT is not ideal. (1) In practice, domain values are often *not* given in ranked lists, and sorting the domains is costly. (2) RankJoinCT invokes procedure check for each tuple in the join result (Example 9); this yields *exponentially* many calls, each taking $O(|\Sigma|(|I_e|^2 + |I_m|))$ time.

6.2 TopKCT: A Brodal Queue Based Algorithm

To remedy the problems of RankJoinCT, we next present TopKCT. In contrast to RankJoinCT, TopKCT does not require ranked lists as input, and invokes check much less.

TopKCT maintains several structures: (1) a heap H_i for each $A_i \in Z$, to store the values in the active domain of A_i; it is able to pop up the top value in H_i in $O(\log |H_i|)$ time, and can be pre-constructed in linear time; (2) a Brodal queue Q, to keep track of tuples to be checked; Q is a worst-case efficient priority queue [6]; it takes $O(1)$ time to insert a tuple and $O(\log |Q|)$ time to pop up the top tuple; and (3) a hash set T to record tuples that were once pushed to Q.

TopKCT is shown in Fig. 5. Its input includes S, $(k, p(\cdot))$, t_e as for RankJoinCT; but instead of ranked lists, it takes m heaps $H_1, \cdots, H_m$ as input. It computes a top-k list T_e like RankJoinCT. The key idea behind TopKCT is that when T_e is nonempty, if t is the next best tuple, then there must exist a tuple $t' \in T_e$ such that t and t' differ in *only one* attribute. Hence it capitalizes on the heaps to pop up a tuple that is guaranteed to be the next best, one at time, rather than to compute costly ranked joins. The tuple is then validated by check, and is added to T_e if it is a candidate target. The process proceeds until either T_e is found or the search space is exhausted, with *the early termination property.*

More specifically, TopKCT first lets T_e be empty (line 1). It then pops the top values out of the m heaps $H_1, \cdots, H_m$ to m vectors $B_1, \cdots, B_m$, respectively (line 2), where B_i is a buffer of the values from H_i for $i \in [1, m]$. Note that $t_Z = (B_1[0], \cdots, B_m[0])$ is the tuple with the highest score. An object o is then formed (lines 3-5), with $2 + m$ members: (1) $o.t$ is a tuple t_0, where $t_0[Z] = t_Z$ and $t_0[R \backslash Z] = t_e[R \backslash Z]$; (2) for $i \in [1, m]$, $o.p_i$ is an integer, initially 0; it is an index of B_i, indicating that $o.t[A_i]$ takes value from $B_i[o.p_i]$; and (3) $o.w$ is a real number, which is the score of $o.t$. The Brodal queue Q is initialized with $\{o\}$ only, and it lets the

hash set $T := \{o.t\}$ (line 6). Note that when Q has multiple objects, it always pops up o' with the highest $o'.w$.

After these, TopKCT populates T_e by iterating the following until T_e has k tuples or Q becomes empty (lines 7-15, where $\|\cdot\|$ denotes cardinality). In each iteration, an object o is popped out from Q (line 8). If $o.t$ is verified a candidate target via check, $o.t$ is added to T_e (line 9). TopKCT then expands Q with tuples that differ from previous ones in *only one* attribute (lines 10-15). To do so, it first expands B_i by adding the top value from H_i to the end of B_i if $o.p_i$ already points to the last value of vector B_i, *i.e.,* all the values in B_i have been inspected (line 11). It then generates a new o' by letting $o.t[A_i] := B_i[o.p_i+1]$, *i.e.,* the value with the highest score next to o (lines 12-14; to simplify the discussion, we assume a weight w_{A_i} for each value when computing $o'.w$, as in Section 3; but this can be lifted). If o' has not been pushed to Q before, *i.e.,* $o' \notin T$, o' is added to Q and T (line 15).

Example 10: Consider the same S and $(2, p(\cdot))$ as in Example 9. Instead of ranked lists, TopKCT takes as input two heaps H_{team} and H_{arena} with the same values as the ranked lists for team and arena, respectively. It first pops the top values Chicago Bulls from H_{team} and United Center from H_{arena}, to form t_0 with $p(t_0) = 2+2 = 4$. It puts t_0 in a Brodal Queue Q. In the first iteration, TopKCT pops the top tuple out of Q (*i.e.,* t_0), and adds it to T_e since t_0 is a valid candidate target. It then pushes t_1 and t_2 to Q, where $t_1[\mathsf{team}, \mathsf{arena}]$ = (Chicago, United Center) and $t_2[\mathsf{team}, \mathsf{arena}]$ = (Chicago Bulls, Chicago Stadium). Then, t_1 (or t_2) is added to T_e as it is valid and is among the first popped from Q. Finally TopKCT returns $T_e = [t_0, t_1]$. □

Analysis. Algorithm TopKCT generates the next best candidate tuple by changing *one* attribute of some tuple already in T_e. As argued earlier, this strategy suffices to find top-k tuples. Better still, TopKCT has the following properties.

Proposition 7: TopKCT *has the early termination property,* i.e., *it stops as soon as T_e is found. In addition, it is instance optimal* w.r.t. *the number of visits of each heap (*pop*'s) among all exact algorithms that use heaps to find top-k candidate targets, with optimality ratio 1.* □

An algorithm A is said to be *instance optimal* if there exist constants c_1 and c_2 such that $cost(A, I) \leq c_1 \cdot cost(A', I) + c_2$ for all instances I and all algorithms A' in the same setting as A, where $cost(A, I)$ is a *cost metric* of A on I [11]. The constant c_1 is called the *optimality ratio*. Here $cost(A, I)$ is the number of pop's performed on each heap by A on I.

Complexity. TopKCT incurs less cost than RankJoinCT. To see this, let n be the maximum size of H_i for $i \in [1, m]$, and the kth tuple in T_e correspond to the Kth tuple in the product of domain values. Then (1) pop of a heap takes at most $O(\log\, n)$ time, and there are at most $K + m$ such operations; (2) there are at most K pop operations on the queue Q, and each takes $O(\log\, Km)$ time; (3) there are at most Km push operations on Q, and each takes $O(1)$ time; and (4) check is invoked K times. Denote the cost of invoking check by c. Putting these together, TopKCT is in $O((K + m)\log\, n + K(m + \log\, K + \log\, m + \mathsf{c}))$ time, in contrast to exponential in K by RankJoinCT.

In light of the inherent intractability, however, K may be an exponential of n in the worst case, *e.g.,* when S does not have k candidate targets; in this case, TopKCT would inevitably exhaust the entire search space. Nevertheless, one can easily verify the following tractable special cases.

Proposition 8: TopKCT *is in* PTIME *when (1) ARs are of form (2) only, or (2) the schema R of I_e is fixed.* □

For instance, in case (1) one can easily see that $K = k$; hence from the analysis above it follows that RJCT is in PTIME, whereas TopKCT still takes exponential time.

As will be experimentally verified shortly, TopKCT actually scales well with real-life data. In addition, by modifying the check step for checking Church-Rosser, TopKCT can also be used to *compute top-k rank joins of unranked lists*.

6.3 TopKCT$_\mathsf{h}$: A Heuristic Algorithm

Finally we outline TopKCT$_\mathsf{h}$, a PTIME heuristic algorithm when all the attributes of R have an infinite domain.

TopKCT$_\mathsf{h}$ first finds a set of k tuples by simply invoking TopKCT without the check step (*i.e.,* line 9 in TopKCT). For each t returned by TopKCT, it greedily revises t with values from I_e and I_m until the revised t is verified a candidate target by check(t, S). It returns the revised tuples as T_e.

TopKCT$_\mathsf{h}$ is in $O((k+m)\log n + k(m+\log k+\log m)+km\mathsf{c})$ time, by the complexity of TopKCT ($k = K$ here), since revising k tuples takes $O(km\mathsf{c})$ time. It is heuristic in nature: while tuples in T_e are guaranteed to be candidate targets of S, they do not necessarily have the highest scores, a tradeoff between the cost and the quality of the solutions.

7. EXPERIMENTAL STUDY

Using both real-life data and synthetic data, we conducted five sets of experiments to evaluate: (1) the effectiveness of algorithm IsCR for deducing target tuples; (2) the effectiveness of RankJoinCT, TopKCT and TopKCT$_\mathsf{h}$ for computing top-k candidate targets; (3) rounds of user interactions; (4) the efficiency of RankJoinCT, TopKCT and TopKCT$_\mathsf{h}$; and (5) the effectiveness of TopKCT when being used for truth discovery, compared with the algorithms of [8, 14].

Experimental setting. Three real-life datasets (Med, CFP and Rest) and synthetic datasets (Syn) were used.

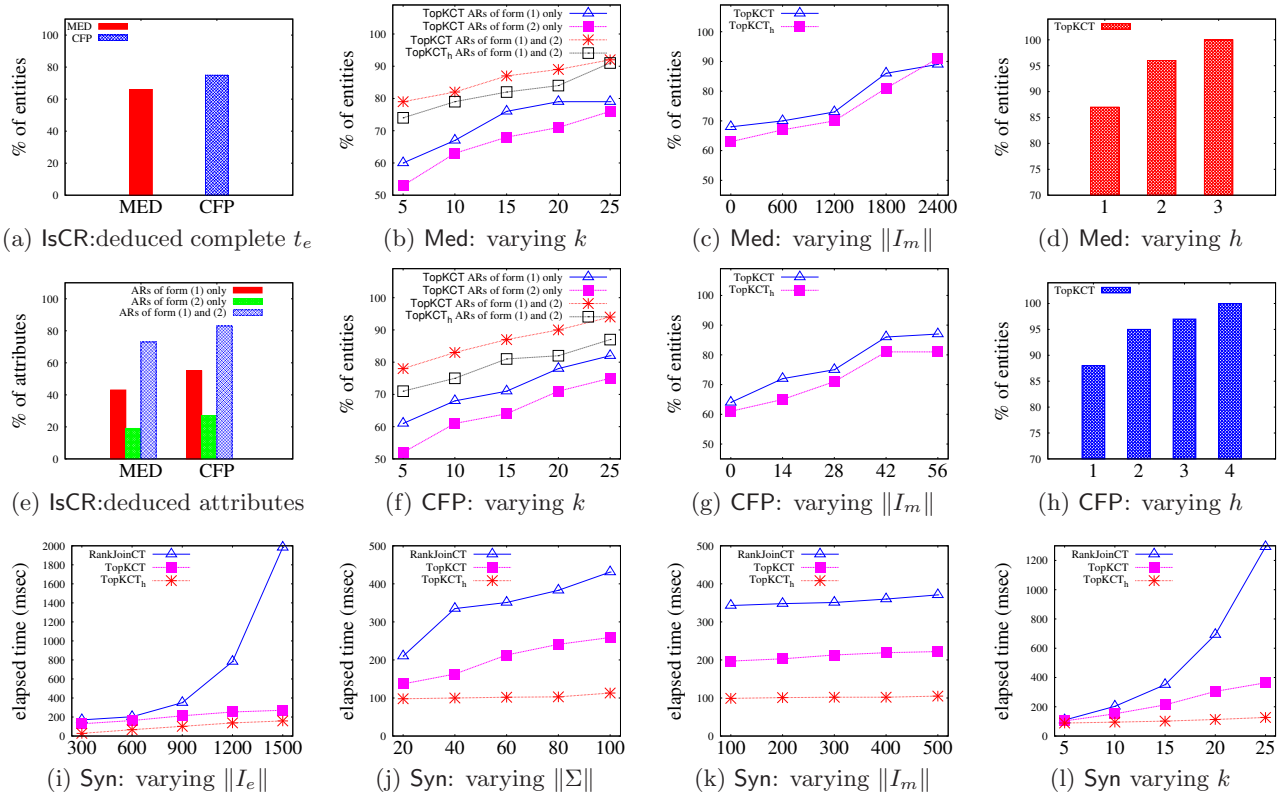
(1) Med was provided by a medicine distribution company (name withheld). It contained sale records of medicines from various stores, specified by a relation schema of 30 attributes such as name, regNo, manufacturer, whose values were not very accurate. Med consisted of 10K tuples for 2.7K entities, where each entity instance ranged from 1 to 83 tuples (4 in average). A set of reference data of 2.4K tuples with 5 attributes was also provided by the company, and we treated it as master data. We manually designed 105 ARs for Med, in which 90 were of form (1) and 15 of form (2).

(2) CFP was extracted from *call for papers/participation* found by Google (*e.g.,* WikiCFP[1], Dbworld[2]). Its attributes included venue, program, and deadline, with values varied in different versions of calls for the same conference. CFP consisted of 100 conferences (entities), with 503 tuples and 22 attributes. The entity instances ranged from 1 to 15 tuples (5 in average). We manually cleaned 55 entries from WikiCFP and treated them as master data, with 17 attributes. We found 43 ARs, with 28 of form (1) and 15 of form (2).

(3) Rest *data*. Rest was the restaurant data used by [8] from `http://lunadong.com/fusionDataSets.htm`. It consisted

[1] http://www.wikicfp.com/cfp/

[2] http://research.cs.wisc.edu/dbworld/browse.html



(a) IsCR:deduced complete t_e (b) Med: varying k (c) Med: varying $\|I_m\|$ (d) Med: varying h

(e) IsCR:deduced attributes (f) CFP: varying k (g) CFP: varying $\|I_m\|$ (h) CFP: varying h

(i) Syn: varying $\|I_e\|$ (j) Syn: varying $\|\Sigma\|$ (k) Syn: varying $\|I_m\|$ (l) Syn varying k

Figure 6: Experimental results

of 8 snapshots of 5149 restaurants in Manhattan, with 246K tuples, crawled from 12 Web sources in one-week intervals. Only the true value of a Boolean attribute closed? was to be determined. We found 131 ARs for Rest, all of form (1).

(4) Syn *data*. We generated a master relation I_m and entity instances I_e of 20 attributes by extending relations stat (Table 1) and nba (Table 2), respectively. The values in I_e and I_m were randomly drawn from the same domains. For its preference model, we assigned random scores to the values in the domains. We also randomly generated a set Σ of 100 ARs, in which 75% were of form (1) and 25% of form (2).

Remark. (1) The specifications for Med, CFP and Rest are Church-Rosser. (2) For preference we counted value occurrences (Section 3). (3) We used $k = 15$ by default in Experiments 1, 2 and 3, and $k = 1$ in Exp-5 (for truth discovery). (4) The ARs for each of the datasets have *similar* structures and often share the same LHS. For each attribute there are typically 3-4 ARs, and the large number of ARs comes from the number of attributes. One can also use profiling methods to *automatically discover* ARs [7, 17] (see Section 4).

Implementation. We implemented the following, all in Python: (1) our algorithms IsCR, RankJoinCT, TopKCT and TopKCT$_h$; (2) a naive algorithm voting based on the preference model that only counts value occurrences, without using ARs; (3) the truth discovery algorithm DeduceOrder of [14] using currency constraints and constant CFDs (see Section 1); and (4) a probabilistic-based truth discovery algorithm copyCEF that utilizes a Bayesian model based on quality measures and copy relationships on data sources [8].

All experiments were conducted on a 64bit Linux Amazon EC2 High-CPU Extra Large Instance with 7 GB of memory, 20 EC2 Compute Units, and 1690 GB of storage. Each experiment was repeated 5 times, and the average is reported here.

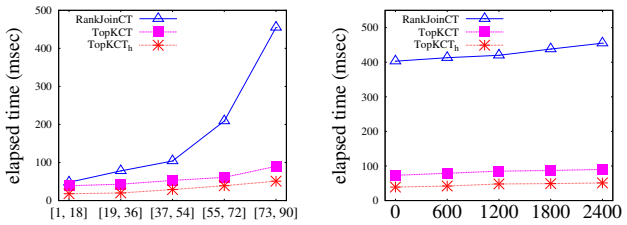
Experimental results. We next present our findings.

Exp-1: Effectiveness of IsCR. Using real-life data Med and CFP, we evaluated the quality of target tuples deduced by IsCR: (a) the percentage of target tuples that were complete; and (b) the percentage of non-null attribute values.

Complete target tuples. Figure 6(a) shows that for 66% of the entities of Med and 72% of CFP, complete target tuples were automatically deduced by IsCR. That is, by leveraging ARs and master data, complete target tuples could be deduced for over 2/3 of the entities *without user interaction*.

Non-null values. Figure 6(e) reports the average percentage of the attributes in Med and CFP for which the most accurate values were deduced. It shows that IsCR found the most accurate values for 42%, 20%, and 73% (resp. 55%, 27%, and 83%) of the attributes in Med (resp. CFP), when ARs of form (1) only, (2) only, and both forms were used, respectively. This tells us the following: (a) IsCR is able to deduce accurate values for a large percentage of attributes, and (b) ARs of forms (1) and (2) interact with each other; indeed, when ARs of both forms were used, the number of accurate values deduced was larger than the *sum* of its counterparts when ARs of form (1) and (2) were used alone. Moreover, when ARs of form (1) or (2) were used only, no complete targets were deduced for Med and CFP (not shown).

Exp-2: Computing top-k candidates. We evaluated the effectiveness of TopKCT (RankJoinCT) and TopKCT$_h$ using Med and CFP. We manually identified the target tuple for each entity, and tested the percentage of entity instances for which the target tuple was among the top-k candidates found by our algorithms. We report the impact of the choice k, the forms of ARs and the size $|I_m|$ of master data on this. Since RankJoinCT and TopKCT are both exact algorithms, the two behaved the same in this set of experiments.



(a) Med: varying $\|I_e\|$ (b) Med: varying $\|I_m\|$

Figure 7: Experimental results

Impact of k. We report the results in Fig. 6(b) (resp. 6(f)) when k was varied from 5 to 25 for Med (resp. CFP). As shown there, (a) The larger k is, the more target tuples are covered by the top-k candidates. (b) To find the target tuples, k does not have to be large. Indeed, when $k = 25$, 92% (resp. 94%) were found by TopKCT and 91% (resp. 87%) by TopKCT$_h$ for Med (resp. CFP). (c) TopKCT did slightly better than TopKCT$_h$ in the quality of candidates found, while TopKCT$_h$ is more efficient than TopKCT (see Exp-4).

Impact of ARs. Figures 6(b) and 6(f) also report the results when Σ consisted of ARs of form (1) only, (2) only, or both. When both forms were used, TopKCT did better than when form (1) or (2) was used alone. In contrast to **Exp-1**, in the latter cases TopKCT could still find many target tuples: for both Med and CFP, it found the targets for 90% of entities when ARs of form (1) or (2) were used only, when $k = 25$.

Impact of $\|I_m\|$. We evaluated the impact of the size of I_m by varying the number of tuples in I_m from 0 to 2400 for Med (resp. 0 to 40 for CFP). As shown in Fig. 6(c) for Med (resp. Fig. 6(g) for CFP), (a) the larger $\|I_m\|$ is, the better TopKCT and TopKCT$_h$ perform, *i.e.,* master data helps improve the quality of top-k candidate targets found by our algorithms; and moreover, (b) even when master data is *unavailable* (*i.e.,* $|I_m| = 0$), TopKCT and TopKCT$_h$ *still work*. Indeed, they were still able to find the target tuples for 63% of Med entities and 64% for CFP, when $k = 15$.

We also tested voting in these settings, and found that voting performed much worse than TopKCT and TopKCT$_h$. It found no more than 50% of target tuples in all the cases.

Exp-3: User interactions. Using Med and CFP, we simulated user interactions as follows. When the deduced target t_e was incomplete, a single attribute B with $t_e[B] =$ null was randomly picked and assigned its accurate value; IsCR and TopKCT were then invoked on the revised t_e. The process repeated until the top-k candidates returned by TopKCT included the target tuple (manually identified, see **Exp-2**).

The results are reported in Figures 6(d) and 6(h) for Med and CFP, respectively, in which the x-axis indicates the number h of interaction rounds, and the y-axis shows the percentage of the target tuples found. The results show that few rounds of interactions are needed to deduce the targets for all the entities: at most 3 for Med, and 4 for CFP.

Exp-4: Efficiency. Using Syn and Med, we evaluated the efficiency of IsCR, RankJoinCT, TopKCT and TopKCT$_h$. To test the impact of I_e, Σ, I_m and k, we set ($\|I_e\|$, $\|I_m\|$, $\|\Sigma\|$, k) = (900, 300, 60, 15) for Syn, and varied one of the four: $\|I_e\|$ from 300 to 1500, $\|\Sigma\|$ from 20 to 100, $\|I_m\|$ from 100 to 500, and k from 5 to 25, while keeping the other three parameters unchanged. For Med we varied $\|I_e\|$ from [1, 18] to [73, 90] and $\|I_m\|$ from 0 to 2400, while keeping $k = 15$ and $\|\Sigma\| = 105$. We found that IsCR took *at most 10ms* in all these cases and hence, do not report it here.

As shown in Figures 6(i) and 7(a), on Med and Syn, (a) all three top-k algorithms are efficient (in less than 2s); (b) TopKCT and TopKCT$_h$ scale well with $\|I_e\|$; and (c) TopKCT outperforms RankJoinCT, and TopKCT$_h$ does better than TopKCT. They behaved consistently when one of $\|\Sigma\|$, $\|I_m\|$ and k was varied, as shown in Figures 6(j), 6(k) and 7(b), and 6(l), respectively. For Syn with $\|I_e\| = 1500$, $\|I_m\| = 300$ and $\|\Sigma\| = 50$, TopKCT$_h$, TopKCT and RankJoinCT took 159ms, 271ms and 1983ms, respectively.

Exp-5: Truth Discovery. Besides determining relative accuracy, we also evaluated the effectiveness of our algorithm TopKCT in truth discovery, against algorithms voting, DeduceOrder [14] and copyCEF [8] on Med, CFP and Rest. Here we used $k = 1$, to find a single target tuple as the true value, in favor of voting, DeduceOrder and copyCEF.

The algorithms were evaluated as follows. (1) We tested voting on Med, CFP and Rest. The results on Med are similar to those on CFP; thus only the results on CFP and Rest are reported here. (2) DeduceOrder was tested on CFP and Rest. For its rules, we extracted all ARs relevant to data currency as currency constraints, and all constant CFDs (which can be expressed as ARs, see Section 2), for each dataset. No such rules were found for Med, and hence only results on CFP and Rest are reported here. (3) We evaluated copyCEF on Rest only, because its required information on data sources is unavailable for Med and CFP. Indeed, Med was from a *single source*, and CFP was crawled from blog posts or Web pages for which the source accuracy could not be determined.

On CFP. On CFP, we tested how many true values (targets) were correctly derived for its entities by the algorithms. We found that voting, DeduceOrder and TopKCT deduced 37%, 0% and 70%, respectively. TopKCT performed almost twice better than voting. While DeduceOrder was not able to find the complete true values for any entity of CFP, it correctly derived 31% of attribute values, which are, however, still much lower than the 83% deduced by IsCR (Fig. 6(e)).

On Rest. On Rest, we evaluated the effectiveness of voting, DeduceOrder, copyCEF and TopKCT. Rest has only one attribute closed? to be determined. Hence we adopted the recall (r), precision (p) and F-measure (F_1) used in [8]: $r = \frac{|G \cap R|}{|G|}$, $p = \frac{|G \cap R|}{|R|}$ and $F_1 = \frac{2pr}{p+r}$, where R is the set of restaurants that were concluded to be closed by an algorithm, and G is the set of restaurants that were indeed closed.

As remarked earlier, our method is complementary to the probabilistic-based truth discovery approaches. Moreover, probabilities derived by these approaches can be incorporated into our model. Indeed, when the probabilities of attribute values returned by copyCEF are set as weights in our preference model, copyCEF can be treated as TopKCT with an empty set of ARs. When the weights in the preference model for TopKCT are set by value occurrences, voting is a special case of TopKCT with an empty set of ARs.

As reported in Table 4, DeduceOrder achieved 100% precision, but was bad on recall (0.15) and F-measure (0.26). While the F-measure of voting was reasonable (0.74), its precision was not very good (0.62). Algorithm copyCEF performed well with balanced precision (0.76) and recall (0.85), and did better in F-measure (0.8) than voting and DeduceOrder. Better still, ARs help here: with a small number of ARs, TopKCT that took value occurrences (like voting) as its preference outperformed copyCEF and voting on F-measure (0.83), and TopKCT that took the possibilities

Method	Prec	Rec	F-msr
DeduceOrder	1.0	0.15	0.26
voting	0.62	0.92	0.74
copyCEF	0.76	0.85	0.8
TopKCT (preference derived by voting)	0.73	0.95	0.82
TopKCT (preference derived by copyCEF)	0.81	0.88	0.85

Table 4: Truth Discovery on Rest

derived by copyCEF further improved copyCEF on precision (0.81) and recall (0.88), even without user interaction.

Observe the following. (1) DeduceOrder did not do well because there was not much currency and consistency information in CFP and Rest that could be utilized by DeduceOrder. Further, the assumption of [14] that the data has to be once correct was too strong for CFP and Rest. These further highlight the need for the study of relative accuracy with ARs. (2) Even without ARs, TopKCT can incorporate the source accuracy (copyCEF) and preference (voting), and performs well in truth discovery. (3) In contrast to Rest, many real-life datasets have a number of attributes that are logically correlated. TopKCT works better on such data than on Rest.

Summary. We find the following. (1) IsCR is effective: it is able to correctly and automatically deduce complete target tuples for at least 66% of the entities, and the most accurate values for 73% of the attributes in the real-life data. (2) ARs of form (1) and form (2) interact with each other and are effective in determining relative accuracy. (3) Our top-k algorithms RankJoinCT and TopKCT (resp. $\mathsf{TopKCT_h}$) are capable of finding the target tuples for 93% (resp. 88%) of entities without user interaction, even when k is small. (4) Few rounds of user interactions are needed for our framework to deduce complete target tuples (3 for Med and 4 for CFP). (5) Our algorithms are efficient: IsCR takes less than 10ms, and $\mathsf{TopKCT_h}$ and TopKCT take 159ms and 271ms respectively, on entity instances consisting of 1500 tuples, I_m of 300 tuples and Σ of 50 ARs. (6) Our method is complementary to truth discovery algorithms, and can incorporate derived probabilities into our preference model. With a small number of ARs, TopKCT improves voting, DeduceOrder and copyCEF in truth discovery, with or without user interactions.

8. CONCLUSION

We have proposed a model for determining relative accuracy for entities in the absence of their true values. We have identified fundamental problems for accuracy, and established their complexity. Based on these, we have introduced a framework for deducing relative accuracy, and provided its underlying algorithms. Our experimental results have verified the effectiveness and efficiency of our techniques.

The study of data accuracy is still in its infancy. We are currently experimenting with large datasets from other domains to evaluate the techniques. We are also studying how to improve the accuracy of data in a database, which is often much larger than entity instances. Furthermore, discovery of ARs deserves a full treatment.

Acknowledgments. Fan and Yu are supported in part by EPSRC EP/J015377/1, the 973 Program 2012CB316200 and NSFC 61133002 of China. Cao is supported in part by the 973 Program 2011CB302602 and NSFC 91118008 of China.

9. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
[2] C. Batini and M. Scannapieco. *Data Quality: Concepts, Methodologies and Techniques*. Springer, 2006.
[3] L. Bertossi, S. Kolahi, and L. Lakshmanan. Data cleaning and query answering with matching dependencies and matching functions. In *ICDT*, 2011.
[4] L. Blanco, V. Crescenzi, P. Merialdo, and P. Papotti. Probabilistic models to reconcile complex data from inaccurate data sources. In *AISE*, 2010.
[5] J. Bleiholder and F. Naumann. Data fusion. *ACM Comput. Surv.*, 41(1), 2008.
[6] G. S. Brodal. Worst-case efficient priority queues. In É. Tardos, editor, *SODA*, pages 52–58. ACM/SIAM, 1996.
[7] F. Chiang and R. Miller. Discovering data quality rules. In *VLDB*, 2008.
[8] X. Dong, L. Berti-Equille, and D. Srivastava. Truth discovery and copying detection in a dynamic world. In *PVLDB*, 2009.
[9] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *TKDE*, 19(1):1–16, 2007.
[10] M. Eppler. *Managing information quality: Increasing the value of information in knowledge-intensive products and processes*. Springer, 2006.
[11] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *JCSS*, 66(4):614–656, 2003.
[12] W. Fan and F. Geerts. *Foundations of Data Quality Management*. Morgan & Claypool Publishers, 2012.
[13] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for capturing data inconsistencies. *TODS*, 33(1), 2008.
[14] W. Fan, F. Geerts, N. Tang, and W. Yu. Inferring data currency and consistency for conflict resolution. In *ICDE*, 2013.
[15] W. Fan, J. Li, S. Ma, N. Tang, and W. Yu. Interaction between record matching and data repairing. In *SIGMOD*, 2011.
[16] C. Fisher, E. Lauria, and C. Matheus. An accuracy metric: Percentages, randomness, and probabilities. *JDIQ*, 1(3), 2009.
[17] P. A. Flach and N. Lachiche. Confirmation-guided discovery of first-order rules with Tertius. *Machine Learning*, 44(3):61–95, 2001.
[18] C. Fox, A. Levitin, and T. Redman. The notion of data and its quality dimensions. *IPM*, 30(1), 1994.
[19] A. Galland, S. Abiteboul, A. Marian, and P. Senellart. Corroborating information from disagreeing views. In *WSDM*, 2010.
[20] I. Gelman. Setting priorities for data accuracy improvements in satisficing decision-making scenarios: A guiding theory. *DSS*, 48(4), 2010.
[21] I. Ilyas, W. Aref, and A. Elmagarmid. Supporting top-k join queries in relational databases. *VLDB. J*, 13(3), 2004.
[22] I. Ilyas, G. Beskales, and M. Soliman. A survey of top-k query processing techniques in relational database systems. *CSUR*, 40(4), 2008.
[23] P. Narman, P. Johnson, M. Ekstedt, M. Chenine, and J. Konig. Enterprise architecture analysis for data accuracy assessments. In *EDOC*, 2009.
[24] F. Naumann and M. Herschel. *An Introduction to Duplicate Detection*. Morgan & Claypool Publishers, 2010.
[25] J. Radcliffe and A. White. Key issues for master data management. Technical report, Gartner, 2008.
[26] K. Schnaitter and N. Polyzotis. Evaluating rank joins with optimal cost. In *PODS*, 2008.
[27] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *CIDR*, 2005.
[28] M. Wu and A. Marian. A framework for corroborating answers from multiple web sources. *IS*, 36(2), 2011.
[29] X. Yin, J. Han, and P. Yu. Truth discovery with multiple conflicting information providers on the Web. *TKDE*, 20(6), 2008.
[30] B. Zhao, B. I. P. Rubinstein, J. Gemmell, and J. Han. A bayesian approach to discovering truth from conflicting sources for data integration. *PVLDB*, 5(6):550–561, 2012.