Tractable Cases of Clean Query Answering under Entity Resolution via Matching Dependencies[⋆]

Jaffer Gardezi
University of Ottawa, SITE.
Ottawa, Canada
jgard082@uottawa.ca

Leopoldo Bertossi
Carleton University, SCS
Ottawa, Canada
bertossi@scs.carleton.ca

Abstract. Matching Dependencies (MDs) are a recent proposal for declarative entity resolution. They are rules that specify, given the similarities satisfied by values in a database, what values should be considered duplicates, and have to be matched. On the basis of a chase-like procedure for MD enforcement, we can obtain clean (duplicate-free) instances; possibly several of them. The clean answers to queries (which we call the resolved answers) are invariant under the resulting class of instances. Identifying the clean versions of a given instance is generally an intractable problem. In this paper, we show that for a certain class of MDs, the characterization of the clean instances is straightforward. This is an important result, because it leads to tractable cases of resolved query answering. Further tractable cases are derived by making connections with tractable cases of CQA.

1 Introduction

For various reasons, such as errors or variations in format, integration of data from different sources, etc., databases may contain different coexisting representations of the same external, real world entity. Those "duplicates" can be entire tuples or values within them. To obtain accurate information, in particular, query answers from the data, those tuples or values should be merged into a single representation.

Identifying and merging duplicates is a process called *entity resolution* (ER) [13, 16]. Matching dependencies (MDs) are a recent proposal for declarative duplicate resolution [17, 18]. An MD expresses, in the form of a rule, that if the values of certain attributes in a pair of tuples are similar, then the values of other attributes in those tuples should be matched (or merged) into a common value.

For example, the MD $R_1[X_1] \approx R_2[X_2] \rightarrow R_1[Y_1] \doteq R_2[Y_2]$ is a symbolic expression saying that, if an R_1-tuple and R_2-tuple have similar values for their attributes X_1, X_2, then their values for attributes Y_1, Y_2 should be made equal. This is a *dynamic* dependency, in the sense that its satisfaction is checked against a pair of instances: the first one where the antecedent holds, and the second one where the identification of values takes place. This semantics of MDs was sketched in [18].

In this paper we use a refinement of that original semantics that was put forth in [23] (cf. also [24]). It improves wrt the latter in that it disallows changes that are irrelevant to the duplicate resolution process. Actually, [23] goes on to define the clean versions of

⋆ Research supported by the NSERC Strategic Network on Business Intelligence (BIN ADC05) and NSERC/IBM CRDPJ/371084-2008.

the original database instance D_0 that contains duplicates. They are called the *resolved instances* (RIs) of D_0 wrt the given set M of matching dependencies. A resolved instance is obtained as the fixed point of a chase-like procedure that starts from D_0 and iteratively applies or enforces the MDs in M. Each step of this chase generates a new instance by making equal the values that are identified as duplicates by the MDs.

In [23] it was shown that resolved instances always exist, and that they have certain desirable properties. For example, the set of allowed changes is just restrictive enough to prevent irrelevant changes, while still guaranteeing existence of resolved instances. The resolved instances that minimize the *overall number* of attribute value changes wrt the original instance are called *minimally resolved instances* (MRIs). On this basis, given a query $\mathcal{Q}$ posed to a database instance D_0 that may contain duplicates, we defined the *resolved answers* wrt Σ as the query answers that are true of all the minimally resolved instances [23].

The concept of resolved query answer has similarities to that of *consistent query answer* (CQA) in a database that fails to satisfy a set of integrity constraints [3, 7, 8]. The consistent answers are invariant under the *repairs* of the original instance. However, data cleaning and CQA are different problems. For the former, we want to compute a clean instance, determined by MDs; for the latter, the goal is obtaining semantically correct query answers. MDs are not (static) ICs. In principle, we could see clean instances as repairs, treating MDs similarly to static FDs. However, the existing repair semantics do not capture the matchings as dictated by MDs (cf. [23, 24] for a more detailed discussion).

The motivation for defining the concept of resolved answers to a query is that even in a database instance containing duplicates, much or most of the data may be duplicate-free. One can therefore obtain useful information from the instance without having to perform data cleaning on the instance. This would be convenient if the user does not want, or cannot afford, to go through a data cleaning process. In other situations the user may not have write access to the data being queried, or any access to the data sources, as in virtual data integration systems [25, 9].

In this paper we show that for a certain sets of MDs whose members depend cyclically on each other, it is possible to characterize the form of the minimally resolved instances for any given instance. In particular, we introduce a recursively defined predicate for identifying the sets of duplicate values within a database instance. This predicate can be combined with a query, opening the ground for tractability via a query rewriting approach to the problem of retrieving the resolved answers to the query.

We also establish connections between the current problem and consistent query answering (CQA) to obtain further tractable cases. When the form of a set of MDs is such that application of one MD cannot affect the application of another MD in the set, the resolved instances of a given database instance are similar to repairs of the instance wrt a set of functional dependencies (FDs). This allows us to apply results on CQA under FDs [15, 21, 30].

This paper is organized as follows. In Section 2 we introduce basic concepts and notation of MDs. In Section 3, we define the important concepts used in this paper, in particular, (minimally) resolved instances and resolved answers to queries. Section 4 contains the main result of this paper, which is a characterization of the minimally resolved instances for certain sets of MDs with cyclic dependency graphs. Section 5 makes some connections with CQA. Section 6 concludes the paper and discusses related and future work. The missing and not already formally published proofs can be found all in [22].

2 Preliminaries

We consider a relational schema $\mathcal{S}$ that includes an enumerable, possibly infinite domain U, and a finite set $\mathcal{R}$ of database predicates. Elements of U are represented by lower case letters near the beginning of the alphabet. $\mathcal{S}$ determines a first-order (FO) language $L(\mathcal{S})$. An instance D for $\mathcal{S}$ is a finite set of ground atoms of the form $R(\bar{a})$, with $R \in \mathcal{R}$, say of arity n, and $\bar{a} \in U^n$. $R(D)$ denotes the extension of R in D. Every predicate $R \in \mathcal{S}$ has a set of attribute, denoted $attr(R)$. As usual, we sometimes refer to attribute A of R by $R[A]$. We assume that all the attributes of a predicate are different, and that we can identify attributes with *positions* in predicates, e.g. $R[i]$, with $1 \leq i \leq n$. If the ith attribute of predicate R is A, for a tuple $t = (c_1, \ldots, c_n) \in R(D)$, $t_R^D[A]$ (usually, simply $t_R[A]$ or $t[A]$ if the instance is understood) denotes the value c_i. For a sequence $\bar{A}$ of attributes in $attr(R)$, $t[\bar{A}]$ denotes the tuple whose entries are the values of the attributes in $\bar{A}$. For a tuple t in a relation instance with attribute A, the pair (t, A) is called a *value position* (usually simply, *position*). In that case, $t[A]$ is the value taken by that position (for the given instance). Attributes have and may share subdomains of U.

In the rest of this section, we summarize some of the assumptions, definitions, notation, and results from [23], that we will need.

We will assume that every relation in an instance has an auxiliary attribute, a surrogate key, holding values that act as *tuple identifiers*. Tuple identifiers are never created, destroyed or changed. They do not appear in MDs, and are used to identify different versions of the same original tuple that result from the matching process. We usually leave them implicit; and "tuple identifier attributes" are commonly left out when specifying a database schema. However, when explicitly represented, they will be the "first" attribute of the relation. For example, if $R \in \mathcal{R}$ is n-ary, $R(t, c_1, \ldots, c_n)$ is a tuple with id t, and is usually written as $R(t, \bar{c})$. We usually use the same symbol for a tuple's identifier as for the tuple itself. Tuple identifiers are unique over the entire instance.[1]

Two instances over the same schema that share the same tuple identifiers are said to be *correlated*. In this case it is possible to unambiguously compare their tuples, and as a result, also the instances.

As expected, some of the attribute domains, say A, have a built-in binary similarity relation $\approx_A$. That is, $\approx_A \subseteq Dom(A) \times Dom(A)$. It is assumed to be reflexive and symmetric. Such a relation can be extended to finite lists of attributes (or domains therefor), componentwise. For single attributes or lists of them, the similarity relation is generically denoted with $\approx$.

A *matching dependency* (MD) [17], involving predicate R, is an expression (or rule), m, of the form

$$m: \ R[\bar{A}] \approx R[\bar{A}] \ \rightarrow \ R[\bar{B}] \doteq R[\bar{B}], \tag{1}$$

[1] An alternative to the use of tuple ids could be the *dynamic mappings* introduced in [28].

with $\bar{A} = (A_1, ..., A_k)$ and $\bar{B} = (B_1, ..., B_{k'})$ lists of attributes from $attr(R)$.[2] We assume the attributes in $\bar{A}$ are all different, similarly for $\bar{B}$. The set of attributes on the left-hand-side (LHS) of the arrow in m is denoted with $LHS(m)$. Similarly for the right-hand-side (RHS). The condition on the LHS of (1) means that, for a pair of tuples t_1, t_2 in (an instance of) R, $t_1[A_i] \approx_i t_2[A_i]$, $1 \leq i \leq k$. Similarly, the expression on the RHS means $t_1[B_i] \doteq t_2[B_i]$, $1 \leq i \leq k'$. Here, $\doteq$ means that the values should be updated to the same value. Accordingly, the intended semantics of (1) is that, for an instance D, if any pair of tuples $t_1, t_2 \in R(D)$ satisfy the similarity conditions on the LHS, then for the same tuples (or tuple ids), the attributes on the RHS have to take the same values [18], possibly through updates that may lead to a new version of D.

Attributes that appear to the right of the arrow in an element of a set M of MDs are called *changeable attributes*. We assume that all sets M of MDs are in *standard form*, i.e. for no two different MDs $m_1, m_2 \in M$, $LHS(m_1) = LHS(m_2)$. All sets of MDs can be put in this form. MDs in a set M can interact in the sense that a matching enforced by one of them may create new similarities that lead to the enforcement of another MD in M. This intuition is captured through the *MD-graph*.

Definition 1. [24] Let M be a set of MDs in standard form. The *MD-graph* of M, denoted $MDG(M)$, is a directed graph with a vertex m for each $m \in M$, and an edge from m to m' iff $RHS(m) \cap LHS(m') \neq \emptyset$.[3] If $MDG(M)$ contains edges, M is called *interacting*. Otherwise, it is called *non-interacting* (NI). □

3 Matching Dependencies and Resolved Answers

Updates as prescribed by an MD m are not arbitrary. The updates based on m have to be justified by m, as captured through the notion of *modifiable value position* in an instance. Values in modifiable positions are the only ones that are allowed to change under a legal update. The notion of modifiable position depends on the syntax of the MDs, but also on the instance at hand on which updates that identify values are to be applied, because the tuple t in a position (t, A) belongs to that instance. We give an example illustrating some issues involved in the definition of modifiability (cf. Definition 2 below).

Example 1. Consider $m\colon R[A] \approx R[A] \rightarrow R[B] \doteq R[B]$ on schema $R[A, B]$, and the instance $R(D)$ shown below.

$R(D)$	A	B
t_1	a_1	c_1
t_2	a_2	c_1
t_3	a_3	c_3
t_4	b_1	c_3
t_5	b_2	c_3

Assume the only non-trivial similarities are $a_1 \approx a_2 \approx a_3$ and $b_1 \approx b_2$. One might be tempted to declare positions (t_i, B) and (t_j, B) as modifiable whenever $t_i[A] \approx t_j[A]$ holds in D. In this case, (t_4, B) and (t_5, B) would be classified as modifiable.

[2] We consider this class to simplify the presentation. However, the results in this paper also apply to the more general case of MDs of the form $R_1[\bar{A}] \approx R_2[\bar{B}] \rightarrow R_1[\bar{C}] \doteq R_2[\bar{D}]$, with the corresponding attributes in $\bar{A}, \bar{B}$ (and in $\bar{C}, \bar{D}$) sharing domains, in particular, similarity relations [22].

[3] That is, they share at least one corresponding pair of attributes.

However, the values in those positions should not be allowed to change, since $t_4[B] = t_5[B]$ and no duplicate resolution is needed. Consequently, we might consider adding the requirement that $t_i[B] \neq t_j[B]$, which would make (t_2, B) and (t_3, B) modifiable, but (t_1, B) non-modifiable.

This is problematic, because a legal update would in general lead to $t_1[B] \neq t_2[B]$ in the new instance (unless the update value for (t_2, B) and (t_3, B) is chosen to be c_1). This would go against the intended meaning of the MD, which tells us that (t_1, B), (t_2, B), and (t_3, B) represent the same entity. As a consequence, $t_1[B] = t_2[B] = t_3[B]$ should hold in the updated instance. $\square$

Example 1 shows that defining modifiability of a value position[4] in terms of just a pair of tuples does not lead to an appropriate restriction on updates. The definition below uses recursion to take larger groups of positions into account.

Definition 2. Let D be an instance, M a set of MDs, and $\mathcal{P}$ be a set of positions (t, G), where t is a tuple of D and G is an attribute of t. (a) For a tuple $t_1 \in R(D)$ and C an attribute of R, the position (t_1, C) is *modifiable wrt* $\mathcal{P}$ if there exist $t_2 \in R(D)$, an $m \in M$ of the form $R[\bar{A}] \approx R[\bar{A}] \ \rightarrow \ R[\bar{B}] \doteq R[\bar{B}]$, and an attribute B of $\bar{B}$, such that $(t_2, B) \in \mathcal{P}$ and one of the following holds:

1. $t_1[\bar{A}] \approx t_2[\bar{A}]$, but $t_1[B] \neq t_2[B]$.
2. $t_1[\bar{A}] \approx t_2[\bar{A}]$ and (t_2, B) is modifiable wrt $\mathcal{P} \smallsetminus \{(t_2, B)\}$.

(b) The position (t_1, B) is *modifiable* if it is modifiable wrt $\mathcal{V} \smallsetminus \{(t_1, B)\}$, where $\mathcal{V}$ is the set of all positions (t, G) with t a tuple of D and G an attribute of t. $\square$

This definition 2 is recursive. The base case occurs when either case 1. applies (with any $\mathcal{P}$) or when there is no tuple/attribute pair in $\mathcal{P}$ that can satisfy part (a). Notice that the recursion must eventually terminate, since the latter condition must be satisfied when $\mathcal{P}$ is empty, and each recursive call reduces the size of $\mathcal{P}$.

Example 2. (example 1 continued) Since $a_2 \approx a_3$ and $c_1 \neq c_3$, (t_2, B) and (t_3, B) are modifiable (base case). Since $a_1 \approx a_2$ and (t_2, B) is modifiable, with case 2. of Definition 2, we obtain that (t_1, B) is also modifiable.

For (t_5, B) to be modifiable, it must be modifiable wrt $\{(t_i, B) \mid 1 \leq i \leq 4\}$, and via t_4. According to case 2. of Definition 2, this requires (t_4, B) to be modifiable wrt $\{(t_i, B) \mid 1 \leq i \leq 3\}$. However, this is not the case since there is no t_i, $1 \leq i \leq 3$, such that $t_4[A] \approx t_i[A]$. Therefore (t_5, B) is not modifiable. A symmetric argument shows that (t_4, B) is not modifiable either.

Notice that the recursive nature of Definition 2 requires defining modifiability in terms of a set of value positions (the set $\mathcal{P}$ in the definition). This set allows us to keep track of positions that have already been "tried". For example, to determine the modifiability of (t_5, B), we must determine whether or not (t_4, B) is modifiable. However, since we have already eliminated (t_5, B) from consideration when deciding about (t_4, B), we avoid an infinite loop. $\square$

[4] Not to be confused with the notion of changeable attribute.

Definition 3. [23] Let D, D' be correlated instances, and M a set of MDs. (D, D') *satisfies* M, denoted $(D, D') \vDash M$, iff: 1. For any pair of tuples $t_1, t_2 \in R(D)$, if there exists an $m \in M$ of the form $R[\bar{A}] \approx R[\bar{A}] \rightarrow R[\bar{B}] \doteq R[\bar{B}]$ and $t_1[\bar{A}] \approx t_2[\bar{A}]$, then for the corresponding tuples (i.e. with same ids) $t'_1, t'_2 \in R(D')$, it holds $t'_1[\bar{B}] = t'_2[\bar{B}]$. 2. For any tuple $t \in R(D)$ and any attribute G of R, if (t, G) is a non-modifiable position, then $t'_R[G] = t_R[G]$. □

Intuitively, D' in Definition 3 is a new version of D that is produced after a single update. Since the update involves matching values (i.e. making them equal), it may produce "duplicate" tuples, i.e. that only differ in their tuple ids. They could possibly be merged into a single tuple in the a data cleaning process. However, we keep the two versions. In particular, D and D' have the same number of tuples. Keeping or eliminating duplicates will not make any important difference in the sense that, given that tuple ids are never updated, two duplicates will evolve in exactly the same way as subsequent updates are performed. Duplicate tuples will never be subsequently "unmerged".

This definition of MD satisfaction departs from [18], which requires that updates preserve similarities. Similarity preservation may force undesirable changes [23]. The existence of the updated instance D' for D is guaranteed [23]. Furthermore, wrt [18], our definition does not allow unnecessary changes from D to D'. Definitions 2 and 3 imply that only values of changeable attributes are subject to updates.

Definition 3 allows us to define a *clean instance* wrt M as the result of a chase-like procedure, each step being satisfaction preserving.

Definition 4. [23] (a) A *resolved instance* (RI) for D wrt M is an instance D', such that there are instances $D_1, D_2, ...D_n$ with: $(D, D_1) \vDash M$, $(D_1, D_2) \vDash M$,..., $(D_{n-1}, D_n) \vDash M$, $(D_n, D') \vDash M$, and $(D', D') \vDash M$. We say D' is *stable*. (b) D' is a *minimally resolved instance* (MRI) for D wrt M if it is a resolved instance and it minimizes the overall number of attribute value changes wrt D. (c) $MRI(D, M)$ denotes the class of MRIs of D wrt M. □

Example 3. Consider the MD $R[A] \approx R[A] \rightarrow R[B] \doteq R[B]$ on predicate R, and the instance D. It has several resolved instances, among them, four that minimize the number of value changes. One of them is D_1. A resolved instance that is not minimal in this sense is D_2.

$R(D)$	A	B
t_1	a_1	c_1
t_2	a_1	c_2
t_3	b_1	c_3
t_4	b_1	c_4

$R(D_1)$	A	B
t_1	a_1	c_1
t_2	a_1	c_1
t_3	b_1	c_3
t_4	b_1	c_3

$R(D_2)$	A	B
t_1	a_1	c_1
t_2	a_1	c_1
t_3	b_1	c_1
t_4	b_1	c_1

□

In this work, as in [23, 24], we are investigating what we could call "the pure case" of MD-based entity resolution. It adheres to the original semantics outlined in [18], which does not specify how the matchings are to be done, but only which values must be made equal. That is, the MDs have implicit existential quantifiers (for the values in common). The semantics we just introduced formally captures this pure case. We find situations like this in other areas of data management, e.g. with referential integrity constraints, tuple-generating dependencies in general [1], schema mappings in data exchange [5], etc. A "non-pure" case, that uses matching functions to realize the matchings as prescribed by MDs, is investigated in [11, 12, 4]. Since there is always an RI [23], there is always an MRI for an instance D wrt M.

The *resolved* answers to a query are *certain* for the class of MRIs for D wrt M.

Definition 5. [23] Let $\mathcal{Q}(\bar{x})$ be a query expressed in the first-order language $L(\mathcal{S})$ associated to schema $\mathcal{S}$ of an instance D. A tuple of constants $\bar{a}$ from U is a *resolved answer* to $\mathcal{Q}(\bar{x})$ wrt the set M of MDs, denoted $D \models_M \mathcal{Q}[\bar{a}]$, iff $D' \models \mathcal{Q}[\bar{a}]$, for every $D' \in \mathit{MRI}(D, M)$. We denote with $\mathit{ResAn}(D, \mathcal{Q}, M)$ the set of resolved answers to $\mathcal{Q}$ from D wrt M. □

Example 4. (example 1 cont.) Since the only MRI for the original instance D is $R(D') = \{\langle t_1, a_1, c_1\rangle, \langle t_2, a_2, c_1\rangle, \langle t_3, a_3, c_1\rangle, \langle t_4, b_1, c_3\rangle, \langle t_5, b_2, c_3\rangle\}$, the resolved answers to the query $\mathcal{Q}(x, y)\colon\ R(x, y)$ are $\{\langle a_1, c_1\rangle, \langle a_2, c_1\rangle, \langle a_3, c_1\rangle, \langle b_1, c_3\rangle, \langle b_2, c_3\rangle\}$. □

For a query $\mathcal{Q}$ and set of MDs M, the *resolved answer problem* is the problem of deciding, given a tuple $\bar{a}$ and instance D, whether or not $\bar{a} \in \mathit{ResAn}(D, \mathcal{Q}, M)$. More precisely, it is defined by

$$RA_{\mathcal{Q},M} := \{(D, \bar{a}) \mid \bar{a} \in \mathit{ResAn}(D, \mathcal{Q}, M)\}. \tag{2}$$

4 Hit-Simple-Cyclic Sets of MDs

In general, the resolved answer decision problem is NP-hard.

Theorem 1. [23] The resolved answer decision problem can be intractable for join-free conjunctive queries and pairs of interacting MDs. More precisely, for the the query $\mathcal{Q}(x, z)\colon\ \exists y R(x, y, z)$, and the following set M of MDs

$$\begin{aligned} m_1 &: R[A] \approx R[A] \rightarrow R[B] \doteq R[B] \\ m_2 &: R[B] \approx R[B] \rightarrow R[C] \doteq R[C] \end{aligned}$$

the resolved answer (decision) problem is *NP*-hard (in data). □

Generally, intractability of the resolved answer problem arises when choices of update values made during one update in the chase sequence can affect subsequent updates. For the case in Theorem 1, when the instance is updated according to m_1, the choice of update values for values in the B column affects subsequent updates made to the values in the C column according to m_2. The resolved answer problem is tractable for non-interacting sets of MDs, because there is no dependence of updates on previous updates.

In this section, we define a class of sets of MDs, called *hit-simple-cyclic* (HSC) sets, for which the resolved answer problem is tractable for an important class of conjunctive queries. Specifically, we introduce a recursively-defined predicate that can be used to identify the sets of values that must be updated to obtain an MRI and the possible values to which they can be updated. For HSC sets, the interaction of the MDs does not lead to intractability, as it is the case for the set of MDs in Theorem 1. This is because the stability requirement of Definition 4 imposes a simple form on MRIs, making it unnecessary to consider the many possible chase sequences.

Definition 6. A set M of MDs is *simple-cycle* (SC) if its MD graph $MDG(M)$ is (just) a cycle, and in all MDs $m \in M$, at most one attribute in $LHS(m)$ is changeable. $\square$

Example 5. For schema $R[A, C, F, G]$, consider the following set M of MDs:

$$m_1\colon\ R[A] \approx R[A] \rightarrow R[C, F, G] \doteq R[C, F, G],$$
$$m_2\colon\ R[C] \approx R[C] \rightarrow R[A, F, G] \doteq R[A, F, G].$$

$MDG(M)$ is a cycle, because attributes in $RHS(m_2)$ appear in $LHS(m_1)$, and vice-versa. Furthermore, M is SC, because $LHS(m_1)$ and $LHS(m_2)$ are singletons. $\square$

SC sets of MDs can be easily found in practical applications. For them, it is easy to characterize the form taken by an MRI.

Example 6. Consider the instance D and a SC set of MDs below, where the only similarities are: $a_i \approx a_j,\ b_i \approx b_j,\ d_i \approx d_j,\ e_i \approx e_j$, with $i, j \in \{1, 2\}$.

$R(D)$	A	B
1	a_1	d_1
2	a_2	e_2
3	b_1	e_1
4	b_2	d_2

$$m_1\colon\ R[A] \approx R[A] \rightarrow R[B] \doteq R[B],$$
$$m_2\colon\ R[B] \approx R[B] \rightarrow R[A] \doteq R[A].$$

If the MDs are applied twice, successively, starting from D, a possible result is:

$R(D)$	A	B
1	a_1	d_1
2	a_2	e_2
3	b_1	e_1
4	b_2	d_2

$\rightarrow$

$R(D_1)$	A	B
1	b_2	d_1
2	a_2	d_1
3	a_2	e_1
4	b_2	e_1

$\rightarrow$

$R(D_2)$	A	B
1	a_2	e_1
2	a_2	d_1
3	b_2	d_1
4	b_2	e_1

It should be clear that, in any sequence of instances $D_1, D_2, \ldots$, obtained from D by applying the MDs, the updated instances must have the following pairs of equal values or matchings (shown through the tuple ids) in Table 1. In any stable instance, the pairs of values in the above tables must be equal. Given the alternating behavior, this can only be the case if all values in A are equal, and similarly for B, which can be achieved with a single update, choosing any value as the common value for each of A and B.

D_i i odd	A	B
tuple (id) pairs	$(1, 4)$, $(2, 3)$	$(1, 2)$, $(3, 4)$

D_i i even	A	B
tuple (id) pairs	$(1, 2)$, $(3, 4)$	$(1, 4)$, $(2, 3)$

Table 1. Table of matchings

In particular, an MRI requires the common value for each attribute to be set to a most common value in the original instance. For D there are 16 MRIs. $\square$

Example 7. (example 5 cont.) The relation R subject to the given M, has two "keys", $R[A]$ and $R[C]$. A relation like this may appear in a database about people: $R[A]$ could be used for the person's name, $R[C]$ the address, and $R[F]$ and $R[G]$ for non-distinguishing information, e.g. gender and age. $\square$

We now define an extension of the class of SC sets of MDs.

Definition 7. A set M of MDs is *hit-simple-cyclic* iff the following hold: (a) In all MDs $m \in M$, at most one attribute in $LHS(m)$ is changeable. (b) Each vertex v_1 in $MDG(M)$ is on at least one cycle, or there is a vertex v_2 on a cycle with at least two vertices such that there is an edge from v_1 to v_2. $\square$

Notice that SC sets are also HSC sets. An example of the MD graph of an HSC set of MDs is shown in Figure 1.

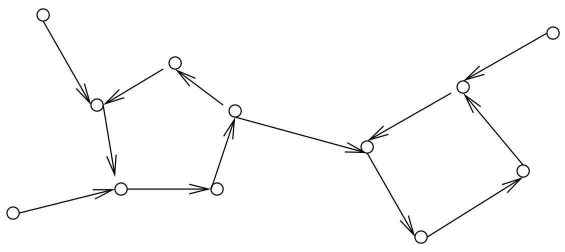

Fig. 1. The MD-graph of an HSC set of MDs

As the previous examples suggest, it is possible to provide a full characterization of the MRIs for an instance subject to an HSC set of MDs, which we do next. For this result, we need a few definitions and notations.

For an SC set M and $m \in M$, if a pair of tuples satisfies the similarity condition of *any* MD in M, then the values of the attributes in $RHS(m)$ must be merged for these tuples. Thus, in Example 6, a pair of tuples satisfying *either* $R[A] \approx R[A]$ *or* $R[B] \approx R[B]$ have *both* their $R[A]$ and $R[B]$ attributes updated to the same value. More generally, for an HSC set M of MDs, and $m \in M$, there is only a *subset* of the MDs such that, if a pair of tuples satisfies the similarity condition of an MD in the subset, then the values of the attributes in $RHS(m)$ must be merged for the pair of tuples. We now formally define this subset.

Definition 8. Let M be a set of MDs, and $m \in M$. (a) The *previous set* of m, denoted $PS(m)$, is the set of all MDs $m' \in M$ with a path in $MDG(M)$ from m' to m. (b) The previous set $PS(M)$ of a set M of MDs is $\bigcup_{m \in M} PS(m)$. $\square$

When applying a set of MDs to an instance, consistency among updates must be enforced. This generally requires computing the transitive closure of similarity relations. For example, suppose both m_1 and m_2 have the conjunct $R[A] \doteq R[A]$. If t_1 and t_2 satisfy the condition of m_1, and t_2 and t_3 satisfy the condition of m_2, then $t_1[A]$ and $t_3[A]$ must be updated to the same value, since updating them to different values would require $t_2[A]$ to be updated to two different values at once. We formally define this transitive closure relation.

Definition 9. Consider an instance D, and set of MDs $M = \{m_1, m_2, \ldots, m_n\}$. (a) For MD $m: \ R[\bar{A}] \approx R[\bar{A}] \rightarrow R[\bar{B}] \doteq R[\bar{B}]$, T_m is the reflexive, symmetric, transitive closure of the binary relation that relates pairs of tuples t_1 and t_2 in D satisfying $t_1[A] \approx t_2[A]$. (b) For a subset M' of M, $T_{M'}$ is the reflexive, symmetric, transitive closure of $\{T_m \mid m \in M'\}$. $\square$

In the case of HSC sets of MDs, the MRIs for a given instance can be characterized simply using the T relation. This result is stated formally below.

Proposition 1. [22] For a set of MDs M and attribute A, let M_A be the set of all $m \in M$ such that $A \in RHS(m)$. For M HSC and D an instance, each MRI for D wrt M is obtained by setting, for each attribute A and each equivalence class E of $T_{PS(M_A)}$, the value of all $t[A]$, $t \in E$, to one of the most frequent values for $t[A]$, $t \in E$. $\square$

Example 8. (example 6 cont.) We represent tuples by their ids. We have:

$$\begin{aligned}
T_{m_1} &= \{(1,2),(3,4),(2,1),(4,3)\} \cup \{(i,i) \mid 1 \le i \le 4\},\\
T_{m_2} &= \{(1,4),(2,3),(4,1),(3,2)\} \cup \{(i,i) \mid 1 \le i \le 4\},\\
T_{\{m_1,m_2\}} &= \{(i,j) \mid 1 \le i,j \le 4\},\\
T_{PS(M_A)} &= T_{PS(m_2)} = T_{\{m_1,m_2\}}, \quad T_{PS(M_B)} = T_{PS(m_1)} = T_{\{m_1,m_2\}}.
\end{aligned}$$

The (single) equivalence class of $T_{PS(M_A)}$ $(T_{PS(M_B)})$ is $\{(i,A) \mid 1 \le i \le 4\}$ $(\{(i,B) \mid 1 \le i \le 4\})$. From Proposition 1, the 16 MRIs are obtained by setting all $R[A]$ and $R[B]$ attribute values to one of the four existing (and, actually, equally frequent) values for them. $\square$

It is possible to prove using Proposition 1 that, for HSC sets, the resolved answer problem is efficiently solvable for join-free conjunctive queries like the one in Theorem 1. In fact, it is shown in [22] that for HSC sets and a significant class of conjunctive queries with restricted joins, the resolved answer problem is solvable in polynomial time using a query rewriting technique.

Definition 10. Let $\mathcal{Q}$ be a conjunctive query without built-ins, and M a set of MDs. $\mathcal{Q}$ is an *unchangeable join* conjunctive query if there are no existentially quantified variables in a join in $\mathcal{Q}$ in the position of a changeable attribute. $UJCQ$ denotes this class of queries. $\square$

Example 9. For schema $\mathcal{S} = \{R[A,B]\}$, let M consist of the single MD $R[A] \approx R[A] \rightarrow R[B] \doteq R[B]$. Attribute B is changeable, and A is unchangeable. The query $\mathcal{Q}_1(x,z):\ \exists y(R(x,y) \wedge R(z,y))$ is not in $UJCQ$, because the bound and repeated variable y is for the changeable attribute B. However, the query $\mathcal{Q}_2(y)$: $\exists x \exists z(R(x,y) \wedge R(x,z))$ is in $UJCQ$: the only bound, repeated variable is x which is for the unchangeable attribute A. If variables x and y are swapped in the first atom of $\mathcal{Q}_2$, the query is not *UJCQ*. $\square$

Theorem 2. [22] For a HSC (or non-interacting) set of MDs M and a $UJCQ$ query $\mathcal{Q}$, there is an effective rewriting $\mathcal{Q}'$ that is efficiently evaluable and returns the resolved answers to $\mathcal{Q}$. $\square$

The rewritten queries of the theorem are expressed in FO logic with an embedded recursively defined predicate that expresses the transitive closure of Definition 9 (e.g. in Datalog) plus a the *count* aggregation operator (of number of different attribute values). They can be evaluated in quadratic time in data [22].

5 A CQA Connection

MDs can be seen as a new form of integrity constraint (IC), with a dynamic semantics. An instance D violates an MD m if there are unresolved duplicates, i.e. tuples t_1 and t_2 in D that satisfy the similarity conditions of m, but differ in value on some pairs of attributes that are expected to be matched according to m. The instances that are consistent with a set of MDs M (or *self-consistent* from the point of view of the dynamic semantics) are resolved instances of themselves with respect to M. Among classical ICs, the closest analogues of MDs are functional dependencies (FDs).

Now, given a database instance D and a set of ICs Σ, possibly not satisfied by D, *consistent query answering* (CQA) is the problem of characterizing and computing the answers to queries $\mathcal{Q}$ that are true in all *repairs* of D, i.e. the instances D' that are consistent with Σ and minimally differ from D [3]. Minimal difference between instances can be defined in different ways. Most of the research in CQA has concentrated on the case of the set-theoretic symmetric difference of instances, as sets of tuples, which in the case of repairs is made minimal under set inclusion, as originally introduced in [3]. Also the minimization of the *cardinality* of this set-difference has been investigated [27, 2]. Other forms of minimization measure the differences in terms of changes of attribute values between D and D' (as opposed to entire tuples) [20, 29, 19, 10], e.g. the number of attribute updates can be used for comparison. Cf. [7, 14, 8] for CQA.

Because of their practical importance, much work on CQA has been done for the case where Σ is a set of functional dependencies (FDs), and in particular for sets, $\mathcal{K}$, of key constraints (KCs) [15, 21, 30], with the distance being the set-theoretic symmetric difference under set inclusion. In this case, on which we concentrate in the rest of this section, a *repair* D' of an instance D becomes a maximal subset of D that satisfies $\mathcal{K}$, i.e. $D' \subseteq D,\ D' \models \mathcal{K}$, and there is no D'' with $D' \subsetneqq D'' \subseteq D$, with $D'' \models \mathcal{K}$ [15].

Accordingly, for a FO query $\mathcal{Q}(\bar{x})$ and a set of KCs $\mathcal{K}$, $\bar{a}$ is a *consistent answer* from D to $\mathcal{Q}(\bar{x})$ wrt $\mathcal{K}$ when $D' \models \mathcal{Q}[\bar{a}]$, for every repair D' of D. For fixed $\mathcal{Q}(\bar{x})$ and $\mathcal{K}$, the *consistent query answering problem* is about deciding membership in the set $CQA_{\mathcal{Q},\mathcal{K}} = \{(D,\bar{a}) \mid \bar{a}$ is a consistent answer from D to $\mathcal{Q}$ wrt $\mathcal{K}\}$.

Notice that this notion of minimality involved in repairs wrt FDs is tuple and set-inclusion oriented, whereas the one that is implicitly related to MDs and MRIs via the matchings (cf. Definition 4) is attribute and cardinality oriented.[5] However, the connection can still be established.

For certain classes of conjunctive queries and ICs consisting of a single KC per relation, *CQA* is tractable. This is the case for the $\mathcal{C}_{forest}$ class of conjunctive queries [21], for which there is a FO rewriting methodology for computing the consistent answers. $\mathcal{C}_{forest}$ excludes repeated relations (self-joins), and allows joins only between non-key and key attributes. Similar results were subsequently proved for a larger class of queries that includes some queries with repeated relations and joins between non-key attributes [30]. The following result allows us to take advantage of tractability results for CQA in our MD setting.

[5] Cf. [23] for a discussion of the differences between FDs and MDs seen as ICs, and their repair processes.

Proposition 2. [22] Let D be a database instance for a single predicate R whose set of attributes is $\bar{A} \cup \bar{B}$, with $\bar{A} \cap \bar{B} = \emptyset$; and m the MD $R[\bar{A}] = R[\bar{A}] \rightarrow R[\bar{B}] \doteq R[\bar{B}]$. There is a polynomial time reduction from $RA_{\mathcal{Q},\{m\}}$ (cf. (2)) to $CQA_{\mathcal{Q},\{\kappa\}}$, where κ is the key constraint $R\colon \bar{A} \rightarrow \bar{B}$. $\square$

This result can be easily generalized to several relations with one such MD defined on each. The reduction takes an instance D for $RA_{\mathcal{Q},\{m\}}$ and produces an instance D' for $CQA_{\mathcal{Q},\{\kappa\}}$. The schema of D' is the same as for D, but the extension of the relation is changed wrt D via counting. Definitions for those aggregations can be inserted into query $\mathcal{Q}$, producing a rewriting Q'. Thus, we obtain:

Theorem 3. [22] Let $\mathcal{S}$ be a schema with $\mathcal{R} = \{R_1[\bar{A}_1, \bar{B}_1], \ldots, R_n[\bar{A}_n, \bar{B}_n]\}$ and $\mathcal{K}$ the set of KCs $\kappa_i : R_i[\bar{A}_i] \rightarrow R_i[\bar{B}_i]$. Let $\mathcal{Q}$ be a FO query for which there is a polynomial-time computable FO rewriting $\mathcal{Q}'$ for computing the consistent answers to $\mathcal{Q}$. Then there is a polynomial-time computable FO query $\mathcal{Q}''$ extended with aggregation[6] for computing the resolved answers to $\mathcal{Q}$ from D wrt the set of MDs $m_i\colon R_i[\bar{A}_i] = R_i[\bar{A}_i] \rightarrow R_i[\bar{B}_i] \doteq R_i[\bar{B}_i]$. $\square$

The aggregation in $\mathcal{Q}''$ in Theorem 3 arises from the *generic* transformation of the instance that is used in the reduction involved in Proposition 2, but here becomes implicit in the query.

This theorem can be applied to decide/compute resolved answers in those cases where a FO rewriting for CQA (aka. consistent rewriting) has been identified.

Example 10. The query $\mathcal{Q}: \exists x \exists y \exists z \exists w(R(x,y,w) \wedge S(y,w,z))$ is in the class $\mathcal{C}_{forest}$ for relational predicates $R[A,B,C]$ and $S[C,E,F]$ and KCs $A \rightarrow BC$ and $CE \rightarrow F$. By Theorem 3 and the results in [21], there is a polynomial-time computable FO query with counting that returns the resolved answers to $\mathcal{Q}$ wrt the MDs $R[A] = R[A] \rightarrow R[B,C] \doteq R[B,C]$ and $S[C,E] = S[C,E] \rightarrow S[F] \doteq S[F]$, namely,

$$\mathcal{Q}''\colon \exists x \exists y \exists z \exists w[R'(x,y,w) \wedge S(y,w,z) \wedge \forall y' \forall w'(R'(x,y',w') \rightarrow \exists z' S(y',w',z')),$$

where $R'(x,y,w) := \exists w' \{R(x,y,w') \wedge \forall y'[\mathit{Count}\{w'' \mid R(x,y',w'')\} \leq$
$\mathit{Count}\{w'' \mid R(x,y,w'')\}]\} \wedge$
$\exists y' \{R(x,y',w) \wedge \forall v[\mathit{Count}\{y'' \mid R(x,y'',v)\} \leq \mathit{Count}\{y'' \mid R(x,y'',w)\}]\}$.

Here, $\mathit{Count}\{x \mid E(x)\}$, for E a first-order expression and x a variable, denotes the number of distinct values of x that satisfy E in the database instance at hand. This "resolved rewriting" wrt the MDs is obtained from the consistent rewriting $\mathcal{Q}'$ for $\mathcal{Q}$ wrt the FDs, by replacing R by R' as indicted above, i.e. using

$$\mathcal{Q}'\colon \exists x \exists y \exists z \exists w[R(x,y,w) \wedge S(y,w,z) \wedge \forall y' \forall w'(R(x,y',w') \rightarrow \exists z' S(y',w',z')),$$

S in $\mathcal{Q}'$ could also be replaced by a similar S' to obtain $\mathcal{Q}''$. However, in this example it is not necessary, because the key values are not changed when the MDs are applied, so the join condition is not affected (the join is on the key values for S, but on the non-key values for R).

[6] This is a proper extension of FO query languages [26, Chapter 8].

Notice that $\mathcal{Q}$ is not in $UJCQ$ because variable y is existentially quantified, participates in a join, and occurs at the position of the changeable attribute $R[B]$ (cf. Definition 10). Therefore, Theorem 2 cannot be used to obtain a query rewriting in this case. $\square$

The example shows that via the CQA connection we obtain rewritable and tractable cases of resolved answering that are different from those provided by Theorem 2.

In this paper we have concentrated on tractable cases of resolved query answering. However, the CQA connection can also be exploited to obtain intractability results, which we briefly illustrate.

Theorem 4. [22] Consider the relational predicate $R[A, B, C]$, the MD $m\colon\ R[A] = R[A] \rightarrow R[B, C] \doteq R[B, C]$, and the query $\mathcal{Q}\colon \exists x \exists y \exists y' \exists z (R(x, y, c) \wedge R(z, y', d) \wedge y = y')$. $RA_{\mathcal{Q},\{m\}}$ is $coNP$-complete (in data). $\square$

This result can be obtained through a reduction and a result in [15, Thm. 3.3]. Notice that the query in Theorem 4 is not $UJCQ$.

6 Conclusions

Matching dependencies first appeared in [17], and their semantics is given in [18]. The original semantics was refined in [11, 12], including the use of *matching functions* for matching two attribute values. An alternative refinement of the semantics, which is the one used in this paper, is given in [23, 24]. Cf. [22] for a thorough complexity analysis, as well as the derivation of a query rewriting algorithm for the resolved answer problem.

This paper builds on the MD-based approach to duplicate resolution introduced in [23]. The latter paper introduced the framework used in this paper, and proved that the resolved answer problem is intractable in some cases. In this paper, we presented a case for which minimally resolved instances can be identified in polynomial time. From this, it follows that the resolved answers can be efficiently retrieved from a dirty database in this case [22]. We also derived other tractable cases using results from CQA.

We used minimal resolved instances (MRIs) as our model of a clean database. Another possibility is to use arbitrary, not necessarily minimal, resolved instances (RIs). This has the advantage of being more flexible in that it takes into account all possible ways of repairing the database. In some cases, the RIs can be characterized by expressing the chase rules in Datalog. Such a direct approach is difficult in the case of MRIs, because of the global nature of the minimality constraint.

References

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

[2] F. Afrati and P. Kolaitis. Repair checking in inconsistent databases: Algorithms and complexity. *Proc. ICDT*, 2009, ACM Press, pp. 31-41.

[3] M. Arenas, L. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. *Proc. PODS*, 1999, ACM Press, pp. 68-79.

[4] Z. Bahmani, L. Bertossi, S. Kolahi and L. Lakshmanan. Declarative entity resolution via matching dependencies and answer set programs. *Proc. KR*, 2012, AAAI Press, pp. 380-390.

[5] P. Barcelo. Logical foundations of relational data exchange. *SIGMOD Record*, 2009, 38(1):49-58.

[6] O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S. Euijong Whang, and J. Widom. Swoosh: A generic approach to entity resolution. *VLDB Journal*, 2009, 18(1):255-276.

[7] L. Bertossi. Consistent query answering in databases. *ACM Sigmod Record*, 2006, 35(2):68-76.

[8] L. Bertossi. *Database Repairing and Consistent Query Answering*, Morgan & Claypool, Synthesis Lectures on Data Management, 2011.

[9] L. Bertossi and L. Bravo. Consistent query answers in virtual data integration systems. In *Inconsistency Tolerance*, Springer LNCS 3300, 2004, pp. 42-83.

[10] L. Bertossi, L. Bravo, E. Franconi, and A. Lopatenko. The complexity and approximation of fixing numerical attributes in databases under integrity constraints. *Information Systems*, 2008, 33(4):407-434.

[11] L. Bertossi, S. Kolahi, and L. Lakshmanan. Data cleaning and query answering with matching dependencies and matching functions. *Proc. ICDT*, 2011, ACM Press.

[12] L. Bertossi, S. Kolahi and L. Lakshmanan. Data cleaning and query answering with matching dependencies and matching functions. *Theory of Computing Systems*, DOI: 10.1007/s00224-012-9402-7, 2012.

[13] J. Bleiholder and F. Naumann. Data fusion. *ACM Computing Surveys*, 2008, 41(1):1-41.

[14] J. Chomicki. Consistent query answering: Five easy pieces. *Proc. ICDT*, 2007, Springer LNCS 4353, pp. 1-17.

[15] J. Chomicki and J. Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Information and Computation*, 2005, 197(1/2):90-121.

[16] A. Elmagarmid, P. Ipeirotis, and V. Verykios. Duplicate record detection: A survey. *IEEE Trans. Knowledge and Data Eng.*, 2007, 19(1):1-16.

[17] W. Fan. Dependencies revisited for improving data quality. *Proc. PODS*, 2008, ACM Press, pp. 159-170.

[18] W. Fan, X. Jia, J. Li, and S. Ma. Reasoning about record matching rules. *Proc. VLDB*, 2009, pp. 407-418.

[19] S. Flesca, F. Furfaro, and F. Parisi. Querying and repairing inconsistent numerical databases. *ACM Trans. Database Syst.*, 2010, 35(2).

[20] E. Franconi, A. Laureti Palma, N. Leone, S. Perri, and F. Scarcello. Census data repair: A challenging application of disjunctive logic programming. *Proc. LPAR*, 2001, pp. 561-578.

[21] A. Fuxman and R. Miller. First-order query rewriting for inconsistent databases. *J. Computer and System Sciences*, 2007, 73(4):610-635.

[22] J. Gardezi, L. Bertossi. Query answering under matching dependencies for data cleaning: Complexity and algorithms. arXiv:1112.5908v1.

[23] J. Gardezi, L. Bertossi, and I. Kiringa. Matching dependencies with arbitrary attribute values: semantics, query answering and integrity constraints. *Proc. Int. WS on Logic in Databases (LID'11)*, ACM Press, 2011, pp. 23-30.

[24] J. Gardezi, L. Bertossi, and I. Kiringa. Matching dependencies: semantics, query answering and integrity constraints. *Frontiers of Computer Science*, Springer, 2012, 6(3):278-292.

[25] M. Lenzerini. Data integration: a theoretical perspective. Proc. PODS 2002, pp. 233-246.

[26] L. Libkin. *Elements of Finite Model Theory*. Springer, 2004.

[27] A. Lopatenko and L. Bertossi. Complexity of consistent query answering in databases under cardinality-based and incremental repair semantics. *Proc. ICDT*, 2007, Springer LNCS 4353, pp. 179-193.

[28] V. Vianu. Dynamic functional dependencies and database aging. *J.ACM*, 1987, 34(1):28-59.

[29] J. Wijsen. Database repairing using updates. *ACM Trans. Database Systems*, 2005, 30(3):722-768.

[30] J. Wijsen. On the first-order expressibility of computing certain answers to conjunctive queries over uncertain databases. *Proc. PODS*, 2010, ACM Press, pp. 179-190.