On the Complexity of Query Answering under Matching Dependencies for Entity Resolution

Leopoldo Bertossi
Carleton University, SCS
Ottawa, Canada

Jaffer Gardezi
University of Ottawa, SITE.
Ottawa, Canada

Abstract. Matching Dependencies (MDs) are a relatively 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; actually possibly several of them. The resolved answers to queries are those that are invariant under the resulting class of resolved instances. In previous work we identified some tractable cases (i.e. for certain classes of queries and MDs) of resolved query answering. In this paper we further investigate the complexity of this problem, identifying some intractable cases. For a special case we obtain a dichotomy complexity result.

1 Introduction

A database may contain several representations of the same external entity. In this sense it contains "duplicates", which is in general considered to be undesirable. And the database has to be cleaned. More precisely, the problem of *duplicate- or entity-resolution* (ER) is about (a) detecting duplicates, and (b) merging duplicate representations into single representations. This is a classic and complex problem in data management, and in data cleaning in particular [9, 11, 4]. In this work we concentrate on the merging part of the problem, in a relational context.

A generic way to approach the problem consists in specifying what attribute values have to be matched (made identical) under what conditions. A declarative language with a precise semantics could be used for this purpose. In this direction, matching dependencies (MDs) have been recently introduced [12]. They represent rules for resolving pairs of duplicate representations (considering two tuples at a time). Actually, when certain similarity relationships between attribute values hold, an MD indicates what attribute values have to be made the same (matched).

Example 1. The similarities of phone and address indicate that the tuples refer to the same person, and the names should be matched. Here, 723-9583 $\approx$ (750) 723-9583 and 10-43 Oak St. $\approx$ 43 Oak St. Ap. 10.

$People\ (P)$	Name	Phone	Address
	John Smith	723-9583	10-43 Oak St.
	J. Smith	(750) 723-9583	43 Oak St. Ap. 10

An MD capturing this cleaning policy, could be the following:

$$P[Phone] \approx P[Phone] \wedge P[Address] \approx P[Address] \ \rightarrow P[Name] \doteq P[Name].$$

This MD involves only one database predicate, but in general, an MD may involve two different relations. □

Here we report on new results (in Section 4) on the computation of resolved query answers wrt. a set of MDs, i.e. of those answers that are invariant under the MD-based ER process. We identify syntactic classes of MDs for which, computing resolved answers to conjunctive queries in a syntactic class, is *always* intractable.

2 Preliminaries

We assume we are dealing with relational schemas and instances. Matching dependencies (MDs) are symbolic rules of the form:

$$\bigwedge_{i,j} R[A_i] \approx_{ij} S[B_j] \;\rightarrow\; \bigwedge_{k,l} R[A_k] \doteq S[B_l], \tag{1}$$

where R, S are relational predicates, and the $A_i, ...$ are attributes for them. The LHS captures similarity conditions on a pair of tuples belonging to the extensions of R and S in an instance D. We abbreviate this formula as: $R[\bar{A}] \approx S[\bar{B}] \;\rightarrow\; R[\bar{C}] \doteq S[\bar{E}]$. MDs have a *dynamic interpretation* requiring that those values on the RHS should be updated to some (unspecified) common value. Those attributes on a RHS of an MD are called *changeable attributes*.

The similarity predicates $\approx$ (there may be more than one in an MD depending on the attributes involved) are treated here as built-ins, but are assumed to satisfy: (a) *symmetry*: if $x \approx y$, then $y \approx x$; and (b) *equality subsumption*: if $x = y$, then $x \approx y$. However, *transitivity* is *not* assumed (and in some application it may not hold).

MDs are to be "applied" iteratively until duplicates are solved. In order to keep track of the changes and comparing tuples and instances, we use global tuple identifiers, a non-changeable surrogate key for each database predicate that has changeable attributes. The auxiliary, extra attribute (when shown) appears as the first attribute in a relation, e.g. t is the identifier in $R(t, \bar{x})$. A *position* is a pair (t, A) with t a tuple id, and A an attribute (of the relation where t is an id). The *position's value*, $t[A]$, is the value for A in tuple (with id) t.

A semantics for MDs acting on database instances was proposed in [13]. It is based on a *chase procedure* that is iteratively applied to the original instance D. A *resolved instance* D' is obtained from a finitely terminating sequence of instances, say

$$D \mapsto D_1 \mapsto D_2 \mapsto \cdots \mapsto D', \tag{2}$$

terminating in D', that satisfies the MDs as *equality generating dependencies* [1], i.e. replacing $\doteq$ by equality.

The semantics specifies the one-step transitions or updates allowed to go from D_{i-1} to D_i, i.e. "$\mapsto$" in (2). Only *modifiable positions* within the instance are allowed to change their values in such a step, and as forced by the MDs. Actually, the modifiable positions syntactically depend on a whole set M of MDs and instance at hand; and can be recursively defined (see [13, 14] for the details). Intuitively, a position (t, A) is modifiable iff: (a) There is a t' such that t and t' satisfy the similarity condition of an MD with A on the RHS; or (b) $t[A]$ has not already been resolved (it is different from one of its other duplicates).

Example 2. Consider the MD $R[A] = R[A] \rightarrow R[B] \doteq R[B]$, and the instance $R(D)$ below. The positions of the underlined values in D are modifiable, because their values are unresolved (wrt the MD).

$R(D)$	A	B
t_1	a	$\underline{b}$
t_2	a	$\underline{c}$

$\mapsto$

$R(D')$	A	B
t_1	a	d
t_2	a	d

D' is a resolved instance since it satisfies the MD interpreted as an FD (the update value d is arbitrary).

D' has no modifiable positions with unresolved values: the values for B are already the same, so there is no reason to change them. □

More formally, the *single step semantics* is a follows. Each pair D_i, D_{i+1} in an update sequence (2), i.e. a chase step, must *satisfy* the set M of MDs, modulo unmodifiability, denoted $(D_i, D_{i+1}) \models_{um} M$, which holds iff: (a) For every MD, say $R[\bar{A}] \approx S[\bar{B}] \rightarrow R[\bar{C}] \doteq S[\bar{D}]$ and pair of tuples t_R and t_S, if $t_R[\bar{A}] \approx t_S[\bar{B}]$ in D_i, then $t_R[\bar{C}] = t_S[\bar{D}]$ in D_{i+1}; and (b) The value of a position can only differ between D_i and D_{i+1} if it is modifiable wrt D_i.

This semantics stays as close as possible to the spirit of the MDs as originally introduced [12], and also *uncommitted* in the sense that the MDs do not specify how the matchings have to be realized.[1]

Example 3. Consider the following instance and set of MDs. Here, attribute $R(C)$ is changeable. Position (t_2, C) is not modifiable wrt. M and D: There is no justification

$R(D)$	A	B	C
t_1	a	b	d
t_2	a	c	$\underline{e}$
t_3	a	b	e

$$R[A] = R[A] \rightarrow R[B] \doteq R[B]$$
$$R[B] = R[B] \rightarrow R[C] \doteq R[C].$$

to change its value *in one step* on the basis of an MD and D. However, position (t_1, C) is modifiable. We obtain two resolved instances for D: D_1 and D_2 below.

$R(D_1)$	A	B	C
t_1	a	b	d
t_2	a	b	d
t_3	a	b	d

$R(D_2)$	A	B	C
t_1	a	b	e
t_2	a	b	e
t_3	a	b	e

D_1 cannot be obtained in a single (one step) update since the underlined value is for a non-modifiable position. However, D_2 can. □

Among the *resolved instances* we prefer those that are closest to the original instance. Accordingly, a *minimally resolved instance* (MRI) of D is a resolved instance D' such that *the number of changes of attribute values* comparing D with D' is a minimum. In Example 3, instance D_2 is an MRI, but not D_1 (2 vs. 3 changes). We denote with $Res(D, M)$ and $MinRes(D, M)$ the classes of resolved, resp. minimally resolved, instances of D wrt M.

Given a conjunctive query $\mathcal{Q}$, a set of MDs M, and an instance D, the *resolved answers* to $\mathcal{Q}$ from D are those that are invariant under the entity resolution process, i.e. they are answers to $\mathcal{Q}$ that are true in all MRIs of D: $ResAns_M(\mathcal{Q}, D) := \{\bar{c} \mid D' \models \mathcal{Q}[\bar{c}],$ for every $D' \in MinRes(D, M)\}$. We denote with $RA(\mathcal{Q}, M)$ the decision problem $\{(D, \bar{c}) \mid \bar{c} \in ResAns_M(\mathcal{Q}, D)\}$.

The definition of resolved answer is reminiscent of that of consistent query answers (CQA) in databases that may not satisfy given integrity constraints (ICs) [2, 5]. Much research in CQA has been about developing (polynomial-time) query rewriting methodologies. The idea is to rewrite a query, say conjunctive, into a new query such that the new query on the inconsistent database returns as usual answers the consistent answers to the original query. In all the cases identified in the literature on CQA (see [6] for a survey, and [17] for recent results) depending on the class of conjunctive query and ICs involved, the rewritings that produce polynomial time CQA have been first-order.

[1] We have proposed and investigated other semantics. One of them is as above, but with a modified chase conditions, e.g. applying one MD at a time. Another one imposes that previous resolutions cannot be unresolved. In [7, 8, 3] a semantics that uses *matching functions* to choose a value for a match is developed.

Doing something similar for resolved query answering (RQA) under MDs brings new challenges: (a) MDs contain the non-transitive similarity predicates. (b) Enforcing consistency of updates requires computing the transitive closure of such operators. (c) The minimality of *value changes* (that is not always used in CQA or considered for consistent rewritings). (d) The semantics of resolved query answering for MD-based entity resolution is given, in the end, in terms of a chase procedure.[2] However, the semantics of CQA is model-theoretic, given in terms repairs that are not operationally defined, but arise from set-theoretic conditions.[3]

3 Tractability and Datalog Query Rewriting

In [14, 15], a query rewriting methodology for RQA under MDs was presented. In this case, the rewritten queries turn out to be Datalog queries with counting, and can be obtained for two main classes of sets of MDs: (a) MDs do not depend on each other, i.e. *non-interacting* sets of MDs [13]; (b) MDs depend cyclically on each other, e.g. a set containing $R[A] \approx R[A] \rightarrow R[B] \doteq R[B]$ and $R[B] \approx R[B] \rightarrow R[A] \doteq R[A]$ (or relationships like this by transitivity).

Here cycles help us, because the termination condition for the chase imposes a simple form on the minimally resolved instances (easier to capture and characterize) [14]. For these sets of MDs a conjunctive query can be rewritten to retrieve, in polynomial time, the resolved answers, provided there are no joins on existentially quantified variables corresponding to changeable attributes: *unchangeable attribute join conjunctive* (UJCQ) queries [15]. For example, for the MD $R[A] = R[A] \rightarrow R[B, C] \doteq R[B, C]$ on schema $R[A, B, C]$, $\mathcal{Q} : \exists x \exists y \exists z (R(x, y, c) \wedge R(z, y, d))$ is *not* UJCQ; whereas $\mathcal{Q}' : \exists x \exists z (R(x, y, z) \wedge R(x, y', z'))$ is UJCQ. For queries outside UJCQ, the resolved answer problem can be intractable even for one MD [15].

The case of a set of MDs consisting of

$$R[A] \approx R[A] \rightarrow R[B] \doteq R[B] \text{ and } R[B] \approx R[B] \rightarrow R[C] \doteq R[C], \qquad (3)$$

which is neither non-interacting nor cyclic, is not covered by the positive cases for Datalog rewriting above. Actually, for this set RQA becomes intractable for very simple queries, like $\mathcal{Q}(x, z) : \exists y R(x, y, z)$, that is UJCQ [13].

4 Intractability of Computing Resolved Query Answers

In the previous section we briefly described classes of queries and MDs for which RQA can be done in polynomial time in data (via the Datalog rewriting). We also showed that there are intractable cases, by pointing to a specific query and set of MDs. The questions that naturally arise are: (a) What happens outside the Datalog rewritable cases in terms of complexity of RQA? (b) The exhibited query and MDs correspond to a more general pattern for which intractability holds? We address these questions here.

For all sets M of MDs we consider below, at most two relational predicates appear in M, and when there are two predicates, both appear in all MDs in M. According to the syntactic restrictions for MDs in (1), those two predicates occur in all conjuncts of an MD in M. Furthermore, all the sets of MDs considered below will turn out to

[2] For some implicit connections between repairs and chase procedures, e.g. as used in data exchange see [16], and as used under database completion with ICs see [10].

[3] For additional discussions of differences and connections between CQA and resolved query answering see [13, 15].

be, as previously announced, both interacting and acyclic. Both notions and others can be captured in terms of the MD *graph*, $MDG(M)$, a directed graph, such that, for $m_1, m_2 \in M$, there is an edge from m_1 to m_2 if there is an overlap between $RHS(m_1)$ and $LHS(m_2)$ (the right- and left-hand sides of the arrows as sets of attributes) [13]. M is acyclic when $MDG(M)$ is acyclic. Our results require several terms and notation that we now define.

Definition 1. Consider a set M of MDs involving the predicates R and S. A *changeable attribute query* $\mathcal{Q}$ is a (conjunctive) query in UJCQ, containing a conjunct of the form $R(\bar{x})$ or $S(\bar{y})$ such that all variables in the conjunct are free and none occur in another conjunct of the form $R(\bar{x})$ or $S(\bar{y})$. Such a conjunct is called a *join-restricted free occurrence* of the predicate R or S. □

By definition, the class of *changeable attribute queries* (CHAQ) is a subclass of UJCQ. Both classes depend on the set of MDs at hand. For example, for the MDs in (3), $\exists y R(x, y, z) \in$ UJCQ $\setminus$ CHAQ, but $\exists w \exists t (R(x, y, z) \wedge S(x, w, t)) \in$ CHAQ. We confine attention to UJCQ and subsets of it because, as mentioned in the previous section, intractability limits the applicability of the duplicate resolution method for queries outside UJCQ. The requirement that the query contains a join-restricted free occurrence of R or S eliminates from consideration certain queries in UJCQ for which the resolved answer problem is trivially tractable. For example, for MDs in (3), the query $\exists y \exists z R(x, y, z)$ is not in CHAQ, but is tractable simply because it does not return the values of a changeable attribute (the resolved answers are the classic answers). The join restriction simplifies the analysis while still including many useful queries.

Definition 2. A set M of MDs is *hard* if for every CHAQ $\mathcal{Q}$, $RA(\mathcal{Q}, M)$ is *NP*-hard. M is *easy* if for every CHAQ $\mathcal{Q}$, $RA(\mathcal{Q}, M)$ is in *PTIME*. □

Of course, a set of MDs may not be hard or easy. In the following we give some syntactic conditions that guarantee hardness for classes of MDs.

Definition 3. Let m be an MD. The symmetric binary relation $LRel(m)$ ($RRel(m)$) relates each pair of attributes A and B such that a conjunct of the form $R[A] \approx S[B]$ (resp. $R[A] \doteq S[B]$) appears in $LHS(m)$ (resp. $RHS(m)$). An *L-component* (*R-component*) of m is an equivalence class of the reflexive and transitive closure, $LRel(m)^{eq}$ (resp. $RRel(m)^{eq}$), of $LRel(m)$ (resp. $RRel(m)$). □

The first results concern *linear pairs* of MDs, i.e. those whose graph $MDG(M)$ consisting of the vertices m_1 and m_2, say

$$m_1\text{: } R[\bar{A}] \approx_1 S[\bar{B}] \rightarrow R[\bar{C}] \doteq S[\bar{E}], \text{ and } m_2\text{: } R[\bar{F}] \approx_2 S[\bar{G}] \rightarrow R[\bar{H}] \doteq S[\bar{I}], \quad (4)$$

with only an edge from m_1 to m_2, i.e. $(R[\bar{C}] \cup S[\bar{E}]) \cap (R[\bar{F}] \cup S[\bar{G}]) \neq \emptyset$, whereas $(R[\bar{H}] \cup S[\bar{I}]) \cap (R[\bar{A}] \cup S[\bar{B}]) = \emptyset$. The linear pair is denoted by (m_1, m_2).

Definition 4. Let (m_1, m_2) be a linear pair as in (4). (a) B_R is a binary (reflexive and symmetric) relation on attributes of R: $(R[U_1], R[U_2]) \in B_R$ iff $R[U_1]$ and $R[U_2]$ are in the same R-component of m_1 or the same L-component of m_2. Similarly for B_S.
(b) An *R-equivalent set* (R-ES) of attributes of (m_1, m_2) is an equivalence class of $TC(B_R)$, the transitive closure of B_R, with at least one attribute in the equivalence class belonging to $LHS(m_2)$. The definition of an *S-equivalent set* (S-ES) is the same, with R replaced by S.
(c) An (R or S)-ES E of (m_1, m_2) is *bound* if $E \cap LHS(m_1)$ is non-empty. □

Theorem 1. Let (m_1, m_2) be a linear pair as in (4), with R and S distinct predicates. Assume that each similarity relation has an infinite set of mutually dissimilar elements. Let E_R and E_S be the classes of R-ESs and S-ESs, resp. The pair (m_1, m_2) is hard if $RHS(m_1) \cap RHS(m_2) = \emptyset$, and at least one of the following *does not* hold:

(a) At least one of the following is true: (i) there are no attributes of R in $RHS(m_1) \cap LHS(m_2)$; (ii) all ESs in E_R are bound; or (iii) for each L-component L of m_1, there is an attribute of R in $L \cap LHS(m_2)$.

(b) At least one of the following is true: (i) there are no attributes of S in $RHS(m_1) \cap LHS(m_2)$; (ii) all ESs in E_S are bound; or (iii) for each L-component L of m_1, there is an attribute of S in $L \cap LHS(m_2)$. □

Theorem 1 says that a linear pair of MDs is hard unless the syntactic form of the MDs is such that there is a certain association between changeable attributes in $LHS(m_2)$ and attributes in $LHS(m_1)$ as specified by conditions (ii) and (iii). When m_1 is applied to an instance, similarities can be produced among the values of attributes of $RHS(m_1)$ which are not required by the chase but result from a particular choice of update values. Such *accidental similarities* affect the subsequent updates made by applying m_2, making the query answering problem intractable [13]. For pairs of MDs satisfying (a)(ii) or (a)(iii) (or (b)(ii) or (b)(iii)) in Theorem 1, the similarities resulting from applying m_2 are restricted to a subset of those that are already present among the values of attributes in $LHS(m_1)$, making the problem tractable.

However, when condition (ii) or (iii) is satisfied, accidental similarities among the values of attributes in $RHS(m_1)$ cannot be passed on to values of attributes in $RHS(m_2)$.

This result gives a syntactic condition for hardness. It is an important result, because it applies to many cases of practical interest. For example, the linear pair (m_1, m_2) in (3) turns out to be hard (for all CHAQ queries, in addition to $\exists y R(x, y, z)$).

All syntactic conditions/constructs on attributes above, in particular, the transitive closures on attributes, are "orthogonal" to semantic properties of the similarity relations. When similarity predicates are transitive, every linear pair not satisfying the hardness criteria of Theorem 1 is easy.

Theorem 2. *(dichotomy for transitive similarity)* Let (m_1, m_2) be a linear pair with $RHS(m_1) \cap RHS(m_2) = \emptyset$. If the similarity operators are transitive, then (m_1, m_2) is either easy or hard. □

The next result concerns *pair-preserving* acyclic sets of MDs, defined by: M is pair-preserving if, for any attribute $R[A]$ occurring in a MD, there is only one attribute $S[B]$ such that $R[A] \approx S[B]$ or $R[A] \doteq S[B]$ occur in an MD. These sets of MDs can be of arbitrary size (still subject to the condition of containing at most two predicates). The pair-preserving assumption typically holds in a duplicate resolution setting, since the values of pairs of attributes are normally compared only if they hold the same type of information (e.g. they are both addresses or both names).

Definition 5. Let M be pair-preserving and acyclic, B an attribute in M, and $M' \subseteq M$. B is *non-inclusive* wrt. M' if, for every $m \in M \setminus M'$ with $B \in RHS(m)$, there is an attribute C such that: (a) $C \in LHS(m)$, (b) $C \notin \bigcup_{m' \in M'} LHS(m')$, and (c) C is *non-inclusive* wrt. M'. □

This is a recursive definition of non-inclusiveness. The base case occurs when C is not in $RHS(m)$ for any m, and so must be inclusive (i.e. not non-inclusive). Because

$C \in LHS(m)$ in the definition, for any m_1 such that $C \in RHS(m_1)$, there is an edge from m_1 to m. Therefore, we are traversing an edge backwards with each recursive step, and the recursion terminates by the acyclicity assumption.

Non-inclusiveness is a generalization of conditions (a) (iii) and (b) (iii) in Theorem 1 to a set of arbitrarily many MDs. It expresses a condition of inclusion of attributes in the left-hand side of one MD in the left-hand side of another. Theorem 3 tells us that a set of MDs that is non-inclusive in this sense is hard. Notice that the condition of Theorem 1 that there exists an ES that is not bound does not appear in Theorem 3. This is because, by the pair-preserving requirement, there cannot be a bound ES for any pair of MDs in the set that is a linear pair. For linear pairs, Theorem 3 becomes Theorem 1.

Theorem 3. Let M be pair-preserving and acyclic. Assume there is $\{m_1, m_2\} \subseteq M$, and attributes $C \in RHS(m_2)$, $B \in RHS(m_1) \bigcap LHS(m_2)$ with: (a) C is non-inclusive wrt $\{m_1, m_2\}$, and (b) B is non-inclusive wrt $\{m_2\}$. Then, M is hard. □

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

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
[2] M. Arenas, L. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. *Proc. PODS*, 1999.
[3] Z. Bahmani, L. Bertossi, S. Kolahi and L. Lakshmanan. Declarative entity resolution via matching dependencies and answer set programs. Proc. KR 2012.
[4] 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.
[5] L. Bertossi. Consistent query answering in databases. *ACM Sigmod Record*, 2006, 35(2):68-76.
[6] L. Bertossi. *Database Repairing and Consistent Query Answering*, Morgan & Claypool, Synthesis Lectures on Data Management, 2011.
[7] L. Bertossi, S. Kolahi, and L. Lakshmanan. Data cleaning and query answering with matching dependencies and matching functions. *Proc. ICDT*, 2011.
[8] L. Bertossi, S. Kolahi and L. Lakshmanan. Data cleaning and query answering with matching dependencies and matching functions. *Theory of Computing Systems*, 2013, 52(3):441-482.
[9] J. Bleiholder and F. Naumann. Data fusion. *ACM Computing Surveys*, 2008, 41(1):1-41.
[10] A. Cali, D. Lembo and R. Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. Proc. PODS 2003, pp. 260-271.
[11] A. Elmagarmid, P. Ipeirotis, and V. Verykios. Duplicate record detection: A survey. *IEEE Trans. Knowledge and Data Eng.*, 2007, 19(1):1-16.
[12] W. Fan, X. Jia, J. Li, and S. Ma. Reasoning about record matching rules. *Proc. VLDB*, 2009.
[13] 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.
[14] J. Gardezi, L. Bertossi. Query rewriting using datalog for duplicate resolution. Proc. 2nd Workshop on the Resurgence of Datalog in Academia and Industry (Datalog 2.0, 2012), Springer LNCS 7494, pp. 86-98, 2012.
[15] J. Gardezi and L. Bertossi. Tractable cases of clean query answering under entity resolution via matching dependencies. Proc. International Conference on Scalable Uncertainty Management (SUM'12), Springer LNAI 7520, pp. 180-193, 2012.
[16] B. ten Cate, G. Fontaine and Ph. Kolaitis. On the data complexity of consistent query answering. Proc. ICDT 2012, pp. 22-33.
[17] J. Wijsen. Certain conjunctive query answering in first-order logic. *ACM Trans. Database Syst.*, 2012, 37(2):9.