Leveraging Probabilistic Existential Rules for Adversarial Deduplication

Jose N. Paredes, Maria Vanina Martinez,
Gerardo I. Simari, and Marcelo A. Falappa

{jose.paredes,mvm,gis,mfalappa}@cs.uns.edu.ar

Dept. of Computer Science and Engineering, Universidad Nacional del Sur (UNS)
Institute for Computer Science and Engineering (UNS–CONICET)
San Andres 800, (8000) Bahia Blanca, Argentina

Abstract. The entity resolution problem in traditional databases, also known as deduplication, seeks to map multiple virtual objects to its corresponding set of real-world entities. Though the problem is challenging, it can be tackled in a variety of ways by means of leveraging several simplifying assumptions, such as the fact that the multiple virtual objects appear as the result of name or attribute ambiguity, clerical errors in data entry or formatting, missing or changing values, or abbreviations. However, in cyber security domains the entity resolution problem takes on a whole different form, since malicious actors that operate in certain environments like hacker forums and markets are highly motivated to remain semi-anonymous—this is because, though they wish to keep their true identities secret from law enforcement, they also have a reputation with their customers. The above simplifying assumptions cannot be made in this setting, and we therefore coin the term "adversarial deduplication". In this paper, we propose the use of probabilistic existential rules (also known as Datalog+/–) to model knowledge engineering solutions to this problem; we show that tuple-generating dependencies can be used to generate probabilistic deduplication hypotheses, and equality-generating dependencies can later be applied to leverage existing data towards grounding such hypotheses. The main advantage with respect to existing deduplication tools is that our model operates under the open-world assumption, and thus is capable of modeling hypotheses over unknown objects, which can later become known if new data becomes available.

1 Introduction

Deduplication of objects in databases, also known as entity resolution, is a classical problem in data cleaning [18]; the basic idea is that databases contain objects that are *potential duplicates*—seemingly different records that correspond to the same entity or object in the real world—that need to be identified and merged [8, 14]. There has been a lot of interest on this topic in the last 20 years, yielding a large body of work that lies mostly in the databases literature. Traditionally, entities are resolved using pairwise similarity over the attributes of reference [18]. The most recent and promising proposal are those based on so-called collective entity resolution [6], which exploits additional relational information in the data since references to different entities may co-occur. For instance, one such approach, called iterative blocking [30], iteratively groups together matching records in blocks, giving rise to the possibility of using the information processed so far to inform further decisions. Data-driven approaches [6, 7, 13] leverage examples to determine if which pairs of records are duplicates. Recently, [3] has defined a declarative framework for entity resolution based on *matching dependencies* (MDs). This formalism, which was first introduced in [16, 17], consists of declarative rules that generalize entity resolution tasks, eliminating duplicates via a matching process. These rules state that certain attribute values in relational tuples, under certain similarity conditions over possibly other attribute values in those tuples, must be made the same. The original semantics for MDs, defined in [17], was redefined and extended in [5].

All of the above-mentioned methods operate under the assumption that the undesired situation that multiple records refer to the same real-world object is caused by a mix of clerical errors in data entry (simple typos), ambiguity in names or other attributes (consider, for instance, the result of transliterations such as “Kurt Gödel” being mapped to “Kurt Goedel”), inconsistent abbreviations and formatting, missing values (for instance, phone number not provided), or changing values (for example, one record has an old phone number while others have an updated one). Though these assumptions adequately reflect many real-world scenarios, we are interested in others in which they cannot be assumed to hold. One example of such a scenario is in cyber-security domains such as malicious hacker forums on the *dark/deep web* (the part that is not directly accessible by traditional browsers and search engines)—the entity resolution problem in this setting becomes quite different. In [26], a system for cyber threat intelligence was developed that gathers data from various social platforms on the Internet, focusing particularly on sites in the dark net and deep net. They collect and store information from hacker forum discussions and marketplaces offering products and services that focus on malicious hacking. The system is designed to extract specific information from marketplaces (regarding sales of malware/exploits) and hacker forums (discussions regarding services and threats). This extracted information is well-structured and can be stored in a relational database; they maintain two databases, one for marketplaces and the other for forums. The crawling and parsing of these sites is carried out periodically so that time-varying data can be collected. For markets, they collect product fields such as item title, item description, vendor name, CVE number (identifier given by the National Vulnerability Database [11, 25]), shipping details, item reviews, etc. For forums, they collect other fields such as topic content, post content, topic author, post author, author status, reputation, and topic interest.

These forums and marketplaces have an interesting characteristic: though malicious actors of course wish to remain anonymous from law enforcement agents that might be patrolling, they also enjoy a reputation with their customers and readers, and so they also wish to remain identifiable. In particular, the same actor typically operates under different screen names, keeping certain aspects constant so that others can recognize them. Furthermore, and perhaps most importantly, they also leave involuntary traces behind that can be leveraged in deduplication efforts. We refer to this as the *adversarial deduplication/entity resolution problem.* The following is a simplified example of the kind of information that we can obtain from dark web forums and marketplaces.

Example 1. As a running example, consider the simplified domain of the malicious hacker forums. Consider the following relational schema regarding information about users and their posts in forums:

$$\mathit{User}(\mathit{Id\_u}, \mathit{Nick}), \mathit{Post}(\mathit{Id\_po}, \mathit{Id\_u}, \mathit{Id\_t}),$$

$$\mathit{HasPosted}(\mathit{Id\_u}, \mathit{Id\_t}),\ \mathit{WritStSim}(\mathit{Id\_u}, \mathit{Id\_u}),$$

which represent users of these forums, posts by users, topics in posts, which users have posted on which topics, and the writing style similarity between users, respectively. Furthermore, *id_u* and *id_po* are the keys for relations *User* and *Post*, respectively. Let D_1 be an instance of this schema, where t_1 is *"private sqli tool"*, t_2 is *"Sql Injection"*, and t_3 is *"trojan servers as a backdoor"*:

$$\begin{aligned} D_1 = \{\ & d_1 : \mathit{Post}(p_1, a_1, t_1), d_2 : \mathit{Post}(p_2, a_2, t_2), d_3 : \mathit{Post}(p_3, a_3, t_3), \\ & d_4 : \mathit{User}(a_1, \text{‘\textit{Blackstar}’}), d_5 : \mathit{User}(a_2, \text{‘\textit{Archer}’}), \\ & d_6 : \mathit{User}(a_3, \text{‘\textit{Angel Eyes}’}), d_7 : \mathit{WritStSim}(a_1, a_2) \\ \}& \end{aligned}$$

■

Finally, another basic assumption generally made by traditional approaches to entity resolution—which can be seen as a consequence of the other assumptions—is that the information contained in the databases is *complete.* In our setting, on the other hand, it may be the case that the relevant information is yet to be discovered; in these settings, the *open-world assumption* becomes crucial, allowing us to entertain deduplication hypotheses over unnamed entities. In the next section, we present a formalism that allows for this kind of reasoning, as well as deal with probabilistic uncertainty.

The rest of this paper is organized as follows: Section 2 presents the extension of Datalog+/− with probabilistic annotations and stratified negation, Section 3 presents a two-stage chase procedure that extends the classical chase with propagation of probabilistic events and EGDs. Sections 4 and 5 introduces deduplication programs and deduplication threshold queries that can be answered via the two-stage chase, respectively. We discuss conclusions and future work in Section 6.

2 Probabilistic Datalog+/− with Stratified Negation

In this section, we present the basics of the language of Datalog+/− from [22], its extension with probabilistic uncertainty from [19], and extend it with stratified negation; we redefine some of the basic concepts to adjust them to our presentation.

We assume the existence of the following pairwise disjoint (countably infinite) sets: a set Δ of constants, a set $\mathbf{V}$ of variables, and a set $\mathcal{N}$ of nulls. Different constants represent different values (unique name assumption), while different nulls may represent the same value. We assume a lexicographic order on $\Delta \cup \mathcal{N}$, with every symbol in $\mathcal{N}$ following all symbols in Δ. We denote by $\mathbf{X}$ sequences of variables $X_1, \ldots, X_k$ with $k > 0$.

Consider relational schemas $\mathcal{R}$ with a possibly infinite data domain Δ, a finite set of database predicates (relations) $R \in \mathcal{R}$, and a set of built-in predicates for evaluating "$=$", "$\neg$", "$\leq$", comparisons, and arithmetic operations. Each relation $R \in \mathcal{R}$ has an associated arity n, thus R has the form $R(A_1, \ldots, A_n)$, where each attribute A_i has associated domain $Dom(A_i) \subseteq \Delta$. For the sake of simplicity we may assume that the A_i's are different, and different predicates do not share attributes. However, different attributes may share the same domain. An atomic formula (or atom) has the form $P(t_1, \ldots, t_n)$, where $t_i \in Dom(A_i) \cup \mathbf{V}$. A conjunction of atoms is often identified with the set of all its atoms. A database instance D for $\mathcal{R}$ is a (possibly infinite) set of ground atoms (or tuples) with predicates from $\mathcal{R}$ and arguments from Δ.

A *homomorphism* is a mapping $\mu : \Delta \cup \mathbf{V} \cup \mathcal{N} \to \Delta \cup \mathbf{V} \cup \mathcal{N}$ such that (i) if $c \in \Delta$ then $\mu(c) = c$, (ii) if $c \in \mathbf{V} \cup \mathcal{N}$ then $\mu(c) \in \Delta \cup \mathcal{N}$, and (iii) μ is naturally extended to atoms, sets of atoms, and conjunctions of atoms, that is, $\mu(p(t_1, \ldots, t_n)) = p(\mu(t_1), \ldots, \mu(t_n))$. A *grounding* is a homomorphism ρ such that $\rho(c) \in \Delta$ for every $c \in \Delta \cup \mathbf{V} \cup \mathcal{N}$. Also, given a conjunction of atoms $\Phi(\mathbf{X})$, we denote with $pos(\Phi(\mathbf{X}))$ the conjunction of atoms in $\Phi(\mathbf{X})$ that appear positive, and with $neg(\Phi(\mathbf{X}))$ the conjunction of atoms in $\Phi(\mathbf{X})$ that appear negated.

TGDs. A *tuple generating dependency* (TGD) is a first-order formula:

$$(\sigma)\, \forall \mathbf{X}\mathbf{Y}\, \Phi(\mathbf{X}, \mathbf{Y}) \to \exists \mathbf{Z} \Psi(\mathbf{X}, \mathbf{Z}) \qquad (1)$$

where $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$ are sequences of variables, $\Phi(\mathbf{X}, \mathbf{Y})$ is a conjunction of atoms and negated atoms over $\mathcal{R}$ (called the *body* of σ, denoted $body(\sigma)$), and $\Psi(\mathbf{X}, \mathbf{Z})$ is a conjunction of atoms over $\mathcal{R}$ (called the *head* of σ, denoted $head(\sigma)$). We use $body^+(\sigma)$ and $body^-(\sigma)$ to denote $pos(\Phi(\mathbf{X}, \mathbf{Y}))$ and $neg(\Phi(\mathbf{X}, \mathbf{Y}))$, respectively. For ease of presentation, we sometimes omit the universal quantifiers, and assume that all variables without quantifier are universally quantified.

We restrict the language so that all the variables in $body^-(\sigma)$ appear already in some atom in $body^+(\sigma)$. Furthermore, we assume Σ is *stratified* [20], *i.e.*, Σ can be partitioned into mutually disjoint sets $\Sigma_1, \ldots, \Sigma_k$ such that:

- If Σ_i contains a predicate A that appears positively in some rule, then there is no rule in $\Sigma_{i+1} \cup \ldots \cup \Sigma_k$ with A in the head.
- If Σ_i contains a predicate A that appears negated in some rule, then there is no rule in $\Sigma_i \cup \ldots \cup \Sigma_k$ with A in the head.

We call $\langle \Sigma_1, \ldots, \Sigma_k \rangle$ a *stratification* of Σ. As in the case of standard TGDs [22], w.l.o.g., we can assume that $\Psi(\mathbf{X}, \mathbf{Z})$ is a single atom (any set not satisfying this condition can be translated to an equivalent set that does).

Definition 1. Let σ be a TGD with $body(\sigma) = \forall \mathbf{X}\mathbf{Y}\, \Phi(\mathbf{X}, \mathbf{Y})$. We say that $body(\sigma)$ is *satisfied* in D through homomorphism h if and only if h is such that $h(pos(\Phi(\mathbf{X}, \mathbf{Y}))) \subseteq D$ and $h(neg(\Phi(\mathbf{X}, \mathbf{Y}))) \nsubseteq D$.

A TGD σ is *satisfied* in D if and only if whenever the body of σ is satisfied through homomorphism h, then there exists an extension h' of h such that $h'(head(\sigma)) \subseteq D$.

EGDs. *Equality generating dependencies* (EGDs) are first order formulas of the form: $\forall \mathbf{X} \Phi(\mathbf{X}) \rightarrow X_i = X_j$, where $\Phi(\mathbf{X})$ is a conjunction of atoms, $\mathbf{X}$ is a sequence of variables, and $X_i, X_j \in \mathbf{X}$. As with TGDs, we sometimes omit the universal quantifiers.

The body of an EGD ϵ, denoted $body(\epsilon)$, corresponds to $\Phi(\mathbf{X})$, and the head, denoted $head(\epsilon)$, corresponds to the equality atom on the right hand side of the rule. An instance D *satisfies* an EGD ϵ, denoted $D \models \epsilon$, if whenever there exists a homomorphism h such that $h(body(\Phi)) \subseteq D$, then it must be the case that $h(X_i) = h(X_j)$. Otherwise, we say that ϵ is *violated* in D.

Probabilistic Annotations

Following [19], we further assume the existence of a finite set of constants Δ_M, a finite set of predicate names $\mathcal{R}_M$ such that $\mathcal{R} \cap \mathcal{R}_M = \emptyset$, and an infinite set of variables $\mathbf{V}_M$, such that $\mathbf{V} \cap \mathbf{V}_M = \emptyset$. These sets give rise to corresponding Herbrand bases and Herbrand universes consisting of all possible ground terms and atoms, respectively; these are denoted with $\mathbf{H}_M$ and $\mathbf{U}_M$, respectively.

Let M be a probabilistic model that represents a joint probability distribution over a (finite) set of random Boolean variables $X = \{X_1, \ldots, X_n\}$ such that there is a 1-to-1 mapping from X to the set of all ground atoms over $\mathbf{H}_M$. Examples of the type of probabilistic models that we assume in this work are Markov Logic [28], Bayesian networks [27], etc.

A (probabilistic) annotation λ, relative to M, is a (finite) set of expressions $\langle A = x_i \rangle$, where A is an atom over $\mathcal{R}_M$, Δ_M, and $\mathbf{V}_M$, $x_i \in \{0, 1\}$, and the X_i's are pairwise distinct. A probabilistic annotation is *valid* if and only if for any two different expressions $A = x, B = y \in \lambda$, there does not exist a substitution θ such that $\theta(A) = \theta(B)$. A *probabilistic scenario* is a valid probabilistic annotation λ for which $|\lambda| = |X|$ and all $\langle A = x_i \rangle$ are such that A is ground. We use $scn(M)$ to denote the set of scenarios in M. A probabilistic annotation is equivalent to a formula that consists of the disjunction of all the scenarios that belong to the set denoted by the annotation. Therefore, with a slight abuse of notation, we sometimes combine annotations with logical operators.

Intuitively, a probabilistic annotation is used to describe an event in which the random variables in a probabilistic model M are compatible with the settings of the random variables described by λ, *i.e.*, each X_i has the value x_i. We will attach probabilistic annotations λ to formulas in our language to produce annotated formulas $F : \lambda$, where F is an atom, a TGD, or an EGD. Intuitively, the annotation means that F holds whenever the event associated with λ occurs. Note that whenever a random variable's value is left unspecified in a probabilistic annotation, the variable is *unconstrained*; in particular, a formula annotated with an empty probabilistic annotation means that the formula holds in every world. An annotated formula $F : \lambda$ is referred to as a *probabilistic atom* whenever F is atom, a *probabilistic TGD* (pTGD) if F is a TGD, and a *probabilistic EGD* (pEGD) whenever F is an EGD.

Definition 2. A probabilistic Datalog+/– knowledge base is a triple $KB = (O, M, af)$, where O is a finite set of atoms, TGDs, and EGDs, M is probabilistic model, and af is an annotation function that maps formulas to probabilistic annotations relative to M.

For ease of presentation, we sometimes use notation "$\sigma : e$" to denote "$af(\sigma) = e$". Given a probabilistic Datalog+/– KB $KB = (O, M, af)$ and a scenario $\lambda \in scn(M)$, O_λ denotes the (non-probabilistic) Datalog+/– knowledge base *induced* by λ. Formally, O_λ consists of all formulas F_i (atoms, TGDs, and EGDs) such that $\lambda_i \subseteq \lambda$ for some $F_i : \lambda_i \in O$.

Example 2. Consider the DB from Example 1, and let $KB_1 = (O_1, M_1, af_1)$ be a Datalog+/– KB. Then, O_1 could contain the following TGD ω_1:

$$(\omega_1)\ User(UID, N) \wedge hasPosted(UID, t_1) \rightarrow hasPosted(UID, t_2)$$

The intuition of this rule is: *"Every User that has posts on topic t_1, also has posts on topic t_2"*. The annotation function af_1 is defined such that $af_1(\omega_1) = e_1(t_1, t_2)$ and $e_1(t_1, t_2)$ is an event associated with the probability that a user posted something related to t_2 given that he has posted something related to t_1.

O_1 could contain the following EGD ϵ_1

$$(\epsilon_1)\ User(UID_1, N) \wedge User(UID_2, N) \rightarrow UID_1 = UID_2.$$

The intuition of this rule is: *"Two users cannot share the same user name."*. This rule is annotated with the empty event, that is $af_1(\epsilon_1) = \emptyset$—therefore, it always holds. ■

Probabilistic model M_1 in the above example can be learned from domain information, and it allows to infer probabilities about possible duplicate user and correlations regarding certain events, such as posts by users given specific topics. We will discuss this in greater detail in Section 4.

3 A Two-stage Chase Procedure

We now extend the classical operational semantics for Datalog+/–, based on the *chase* procedure, for the case of the extensions discussed above. This procedure provides the semantics for the application of rules and constraints, extending the well-known (oblivious) chase algorithm for testing implications of data dependencies [4, 23] or completing deductive databases [1]. The chase provides a mechanism for repairing a database instance relative to a set of dependencies, so that the result of the chase (a database instance) satisfies the dependencies. Note that, though [19] presents a chase procedure for probabilistic Datalog+/– ontologies, our two-stage chase includes stratified negation and EGDs.

The building block for the chase procedure is the TGD *chase rule*; in our setting, we must also deal with the propagation of probabilistic annotations.

pTGD Chase Rule. We will now define the probabilistic TGD chase rule for a knowledge base $KB = (O = (D, \Sigma), M, af)$. Let $\sigma \in O$ be a pTGD of the form $\forall \mathbf{X}\mathbf{Y}\, \Phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z} \Psi(\mathbf{X}, \mathbf{Z}) : e_\sigma$; then, σ is *applicable* to D if and only if there exists a homomorphism h such that one the following cases occur:

a) 1) h maps the atoms in $pos(\Phi(\mathbf{X}, \mathbf{Y}))$ and $neg(\Phi(\mathbf{X}, \mathbf{Y}))$ to probabilistic atoms $\{\alpha_1 : e_1, ..., \alpha_k : e_k\} \subseteq D$ and $\{\alpha_{k+1} : e_{k+1}, ..., \alpha_j : e_j\} \nsubseteq D$, and
   2) $h(e_\sigma \wedge \bigwedge_{i=1,...,k} e_i)$ is a valid probabilistic annotation.

b) 1) h maps the atoms in $pos(\Phi(\mathbf{X}, \mathbf{Y}))$ and $neg(\Phi(\mathbf{X}, \mathbf{Y}))$ to probabilistic atoms $\{\alpha_1 : e_1, ..., \alpha_k : e_k\} \subseteq D$ and $\{\alpha_{k+1} : e_{k+1}, ..., \alpha_j : e_j\} \subseteq D$, and
   2) $h(e_\sigma \wedge \bigwedge_{i=1,...,k} e_i \wedge \bigwedge_{i=k+1,...,j} \neg e_i)$ is a valid probabilistic annotations.

Let σ be applicable to D, and h' be a homomorphism that extends h as follows: for each term $X_i \in \mathbf{X}$, $h'(X_i) = h(X_i)$ and $h'(Z) = z_j$, where Z is existentially quantified in *head*(σ) and z_j is a "fresh" null, *i.e.*, $z_j \in \Delta_N$, z_j does not occur in D, and z_j lexicographically follows all other nulls already introduced. The application of σ on D adds to D one of the following probabilistic atoms according to the case where σ is applicable:

a) $h'(\Psi(\mathbf{X}, \mathbf{Z})) : h'(e_\sigma \wedge \bigwedge_{i=1,...,k} e_i)$.

b) $h'(\Psi(\mathbf{X}, \mathbf{Z})) : h'(e_\sigma \wedge \bigwedge_{i=1,...,k} e_i \wedge \bigwedge_{i=k+1,...,j} \neg e_i)$.

As a second step, we apply the equality-generating dependencies. We do this in a separate step over the result of the pTGD chase in order to avoid undesired interactions between TGDs and EGDs that can lead to undecidability [22], since the standard way to avoid this via separability is too strong an assumption in our case (see Section 4).

pEGD Chase Rule. Let $KB = (O = (D, \Sigma), M, af)$ be a probabilistic Datalog+/– KB, and $\epsilon \in \Sigma$ be a pEGD of the form $\forall \mathbf{X}\, \Phi(\mathbf{X}) \rightarrow X_m = X_n : e_\epsilon$. Then, ϵ is *applicable* to D if and only if there exists a homomorphism h such that:

a) h maps the atoms in $\Phi(\mathbf{X})$ to probabilistic atoms $\{\alpha_1 : e_1, ..., \alpha_k : e_k\} \subseteq D$, and

b) $h(e_\epsilon \wedge \bigwedge_{i=1,...,k} e_i)$ is a valid probabilistic annotation.

Let ϵ be applicable to D, and $\Phi_{X_m} = \{\alpha_1 : e_1, ..., \alpha_l : e_l\} \subseteq \{\alpha_1 : e_1, ..., \alpha_k : e_k\}$ be the set of probabilistic atoms where X_m occurs in $\alpha_{1 \leq i \leq l}$; then, the application of ϵ on D results in:

(i) The chase *fails* if $h(X_m)$ and $h(X_n)$ are constants; this is called a *hard violation*. Otherwise,
(ii) for all $\alpha_i : e_i \in \Phi_{X_m}$ replace $h(X_m)$ with $h(X_n)$ and e_i with $h(e_\epsilon \wedge \bigwedge_{i=1,\ldots,k} e_i)$, if $h(X_m)$ is a null and $h(X_n)$ is a constant, or $h(X_m)$ precedes $h(X_n)$ in the lexicographic order.

The following section shows how this chase procedure can be used to answer probabilistic deduplication queries.

4 Deduplication Programs

In this section we define a subclass of Probabilistic Datalog+/− programs that are specifically tailored to declaratively capture the requirements for a deduplication task in an open-world, adversarial environment.

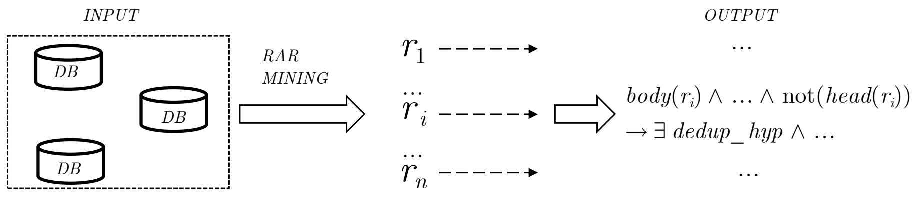
Towards this end, we assume that there exists a set $\mathcal{E} \subseteq \mathcal{R}$ that we identify as *entity relations*, *i.e.*, relations that represent entities of interest in the application domain. Tuples in these relations have identifiers—keys—that allows us to compare extensions of the same predicate in different instances and to trace changes to them. Tuple identifiers can be accommodated by simply adding to each predicate $E \in \mathcal{E}$ an extra attribute (usually denoted with T) that acts as a primary key in E. Finally, there is also a predefined predicate called *dedup_hypth*.

Our formalism is inspired in (relational) *matching dependencies* (RMDs) [5], which provide a declarative way of defining conditions under which records (or attribute values) should be made equal—they also establish how this matching is supposed to be done by means of *matching functions*. However, as we discussed before, RMDs are developed under a closed-world assumption, *i.e.*, everything that is not contained in the database instance is assumed to be false (or to not exist). Nevertheless, we argue that in the type of domain applications we are interested in, this assumption is not realistic and therefore we must agree in that the database instances we are dealing with are potentially incomplete. Under such scenarios, RMDs are not expressive enough to capture all potential matchings, since there might be matchings for which we cannot establish specific conditions to find them but rather just make *hypotheses* based on (weaker) evidence of other nature—the possibility of *value invention* afforded by existential rules thus becomes crucial.

To specify the kind of rules we require, we slightly modify pTGDs so they can be used to infer potential matchings (or *matching hypotheses*). For this, we assume that for every attribute A_j appearing in a n-nary relation $R \in \mathcal{R}$, where the schema of R is $R(A_1, \ldots, A_n)$, there exists a binary similarity relation $\approx_{A_j}$ that is reflexive and symmetric. Thus, they are capable of expressing similarity conditions over attribute values as done in RMDs, since the "$\approx$" predicates can simply occur in the body.

dhTGDs. A *Duplication Hypothesis Tuple Generating Dependency* (or dhTGD) σ is simply a pTGD of the following form:

$$\forall \mathbf{X}\mathbf{Y}\, \Phi(\mathbf{X},\mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \mathit{dedup\_hypth}(X, Z) \wedge \Gamma(\mathbf{X},\mathbf{Z})$$



INPUT
DB
DB
DB
RAR MINING
r_1
...
r_i
...
r_n
OUTPUT
...
$body(r_i) \wedge \ldots \wedge$ not$(head(r_i))$
$\rightarrow \exists\ dedup\_hyp \wedge \ldots$
...

Fig. 1. Overview of a possible dhTGD learning process: relational association rule mining algorithms are applied over a set of available data sources; each such rule then leads to one output dhTGD characterizing situations that violate the corresponding association rule and therefore are the basis of a duplicate entity hypothesis.

where $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$ are sequences of variables, X and Z are variables and $X \in \mathbf{X}, Z \in \mathbf{Z}$, $\Phi(\mathbf{X}, \mathbf{Y})$ is a conjunction of atoms and negated atoms over $\mathcal{R}$, and Γ is also a conjunction of atoms.

dhTGDs are a special class of pTGDs that generate *deduplication hypotheses*—essentially, they allow to create hypotheses about potential matchings, which are kept and later confirmed or discarded whenever *new information* is added to (or extracted from) the data sources. Informally, dhTGDs capture situations in which we have reasons to believe that entities may be duplicated but we do not know *which* other entity could correspond to them. Figure 1 describes one possible way in which dhTGDs could be automatically derived from data. A relational association rule mining algorithm (such as [12]) is used to discover patterns in the available data sources, which are rules of the form $r : body(r) \rightarrow head(r))$; therefore, entities that satisfy the conditions in the body but not those in the head are potential candidates for entities that have duplicate identities—see Example 3 below for a concrete scenario.

pEGDs as companions to dhTGDs. As we saw in Section 3, equality-generating dependencies are applied as a second step after all (dh)TGDs are applied; the goal of this step is to complete the database with the information that might be available with respect to the null values in the *dedup_hypth* atoms. One kind of pEGDs that might be useful can be automatically derived from dhTGDs σ by taking the negation of $body^{-}(\sigma)$ as the body of the pEGD. Figure 2 illustrates this procedure, and the following is a simple example based on the running example.

Example 3. Consider the running example, and suppose we have obtained a relational association rule that states that "if a user has posts related to topic t_1, then he has posts related to topic t_2". This can be translated into the following dhTGD:

$$(\omega_2)\ User(UID, N) \wedge hasPosted(UID, t_1) \wedge not(hasPosted(UID, t_2)) \rightarrow \\ \exists Z\ dedup\_hypth(UID, Z) \wedge hasPosted(Z, t_2).$$

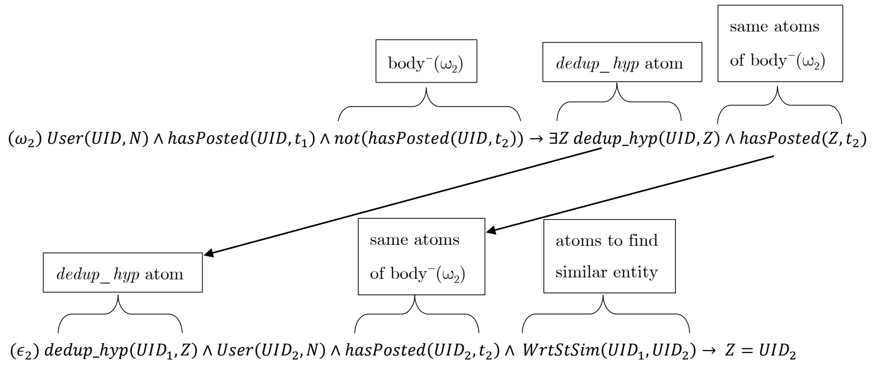

body⁻(ω₂)
dedup_hyp atom
same atoms of body⁻(ω₂)
(ω₂) User(UID, N) ∧ hasPosted(UID, t₁) ∧ not(hasPosted(UID, t₂)) → ∃Z dedup_hyp(UID, Z) ∧ hasPosted(Z, t₂)
dedup_hyp atom
same atoms of body⁻(ω₂)
atoms to find similar entity
(ϵ₂) dedup_hyp(UID₁, Z) ∧ User(UID₂, N) ∧ hasPosted(UID₂, t₂) ∧ WrtStSim(UID₁, UID₂) → Z = UID₂

Fig. 2. Illustration of one possible procedure with which a pEGD can be derived from an existing dhTGD (σ)—the (unnegated) atoms from $body^{-}(\sigma)$ plus additional conditions that help find candidate entities to instantiate the deduplication hypothesis.

The intuition of this rule is *"If a user that has some post on topic t_1 but he does not have any post on topic t_2, then we generate the hypothesis that there exists another user corresponding to him (whose identity is currently unknown) who has posted on topic t_2"*.

As shown in Figure 2, an equality-generating dependency can be derived from ω_2 by taking the atoms from $body^{-}(\omega_2)$ and some additional conditions that, when true, can help obtain the identity of the hitherto unknown entity that participates in the deduplication hypothesis:

$$(\epsilon_2)\ \mathit{dedup\_hypth}(\mathit{UID}_1, Z) \wedge \mathit{User}(\mathit{UID}_2, N) \wedge \mathit{hasPosted}(\mathit{UID}_2, t_2) \wedge \mathit{WritStSim}(\mathit{UID}_1, \mathit{UID}_2) \rightarrow Z = \mathit{UID}_2.$$

The intuition of this rule is *"If we have a duplicate user hypothesis between user UID_1 and an unknown user Z, and user UID_2 has posts on topic t_2 with similar writing style to user UID_1, then Z may correspond to UID_2"*. ■

Putting all of this together, the following example illustrates how the chase procedure unfolds over the scenario in the running example.

Example 4. Considering the DB D_1 presented in Example 1, a probabilistic Extended Datalog+/− KB representing a deduplication program can be $KB_2 = (O_2, M, \mathit{af})$ where $O_2 = (D_1, \Sigma_2)$ and M_1 is the same probabilistic model from Example 2. The set Σ_2 contains the (classical) TGD:

$$(\sigma)\ \mathit{User}(\mathit{UID}, N) \wedge \mathit{Post}(\mathit{Id\_po}, \mathit{UID}, T) \rightarrow \mathit{hasPosted}(\mathit{UID}, T),$$

and also contains the dhTGD ω_2 and pEGD ϵ_2 from the previous example. The annotation function af is defined as follows:

- $\mathit{af}(\omega_2) = e_1(t_1, t_2)$, where the event $e_1(t_1, t_2)$ is associated with the probability that a user has some post on t_2 given that he has some post on t_1.

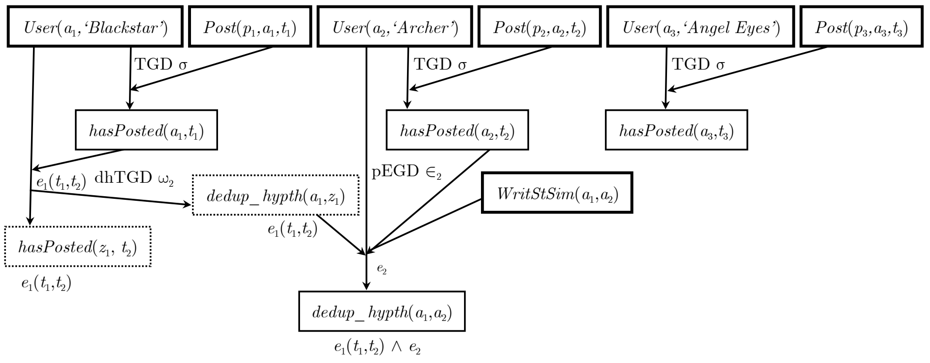


Fig. 3. Graphic representation of the chase for Example 4. Boxes with bold outline represent atoms in D_1, boxes with dashed outline represent atoms that contain null values, and boxes with standard outline represent atoms without nulls obtained from some dhTGD or pEGD. The events associated with each atom are represented underneath the corresponding box; all atoms in D_1 are associated with empty events, and the events propagated are represented next to the arrow of the corresponding dhTGD or pEGD.

λ_i	$e_1(t_1, t_2)$	e_2	Probability
λ_1	*true*	*true*	0.76
λ_2	*true*	*false*	0.19
λ_3	*false*	*true*	0.04
λ_4	*false*	*false*	0.01

Fig. 4. Joint probability distribution over the events $e_1(t_1, t_2)$ and e_2 used in the running example. In general, such distributions can be compactly specified via a probabilistic model like a Bayesian network.

– $af(\epsilon_2) = e_2$, where the event e_2 is associated with the probability that two users with similar writing style actually correspond to the same underlying user.

Figure 3 illustrates the chase for this example. Note that dhTGD ω_2 is applicable because $body^+(\omega_2)$ is mapped to D_1 and $body^-(\omega_2)$ is not mapped to D_1; also note how the event $e_1(t_1, t_2)$ is propagated to the generated atoms. Then, pEGD ϵ_2 is applicable, so null value z_1 is instantiated and the events $e_1(t_1, t_2)$ and e_2 are propagated. Finally, for simplicity we specify M directly as a joint probability distribution, considering all possible worlds λ_i for the events $e_1(t_1, t_2)$ and e_2 (cf. Figure 4). ■

In the previous example, that the application of the dhTGD ω_2 produces a deduplication hypothesis for user a_2, but the lack of information causes the candidate for such a duplicate to remain as a null value. However, the subsequent application of pEGD ϵ_2 lets us conclude that user a_2—who has a similar writing

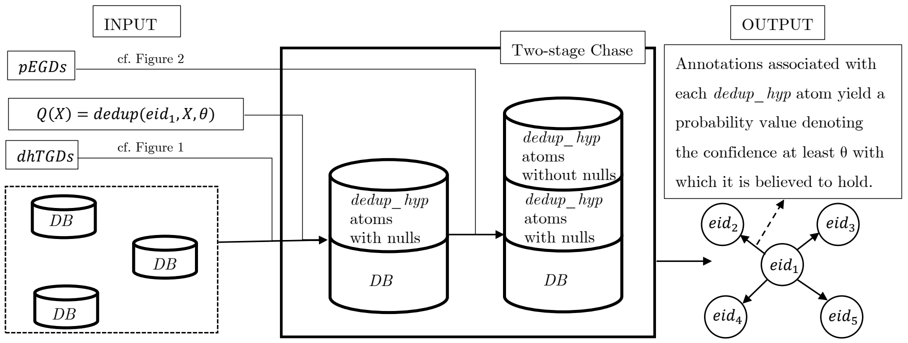

INPUT
pEGDs
cf. Figure 2
$Q(X) = dedup(eid_1, X, \theta)$
dhTGDs
cf. Figure 1
DB
DB
DB
Two-stage Chase
dedup_hyp atoms with nulls
DB
dedup_hyp atoms without nulls
dedup_hyp atoms with nulls
DB
OUTPUT
Annotations associated with each dedup_hyp atom yield a probability value denoting the confidence at least θ with which it is believed to hold.
eid_1
eid_2
eid_3
eid_4
eid_5

Fig. 5. Overview of the application of probabilistic Datalog+/– to answer deduplication queries. A set of input databases is considered along with a set of dhTGDs. In the first stage of the chase procedure, a set of *dedup_hypth* atoms is generated; in the second stage, a set of pEGDs is used to assign concrete values to nulls in such atoms—both stages involve keeping track of probabilistic events stating conditions under which inferences hold. The output is a set of *dedup_hypth* atoms with probability values, which can be used to answer the input query.

style to a_1—is a likely option (event $e_1(t_1, t_2) \wedge e_2$ has probability 0.76 according to Figure 4). Note that the dhTGD could be applied multiple times, so that if there are other possible candidates, fresh *dedup_hypth* atoms can be generated as needed.

5 Finding Duplicates: Threshold Deduplication Queries

The problem of identifying seemingly different entities within a knowledge base (or set of knowledge bases) that actually correspond to the same *real world* entity can be addressed as a query answering problem: given a knowledge base, we want to have a principled way of answering questions regarding the occurrence of duplicate entities. The chase procedure presented above can be used to answer queries of the form:

$$Q(X) = dedup(eid, X, \theta),$$

where *eid* is a specific entity of interest, and $\theta \in [0, 1]$ represents a threshold probability value. Answers to queries of this form simply consist of possible values for variable X such that the chase procedure generates atoms $dedup\_hypth(eid, val)$ with an associated event e with $Pr_M(e) \geq \theta$. Figure 5 illustrates the overall query answering process.

The following result states that the two-stage chase procedure described in Section 3 can be stopped after a finite number of steps in order to answer threshold deduplication queries whenever the underlying classical TGDs belong to a fragment of Datalog+/– for which conjunctive queries are decidable; this can be guaranteed via syntactic conditions such such as *guardedness* [22], *stickiness* [10], *acyclicity*, or their weak counterparts [9, 10, 15], or semantic conditions such as *shyness* [21], finite expansion sets (*fes*), bounded treewidth sets (*bts*), and finite unification sets (*fus*)) [2]—cf. [29, Section 1.3.3] for a brief overview of many such conditions.

Proposition 1. Let $KB = (O = (D, \Sigma), M, \mathit{af})$ be a probabilistic Datalog+/– ontology with stratified negation, and $Q(X) = \mathit{dedup}(\mathit{eid}, X, \theta)$ be a threshold deduplication query. If O belongs to a fragment of Datalog+/– for which a finite portion of $\mathit{chase}(D, \Sigma)$ can be computed in order to answer conjunctive queries, then there exists a finite portion of the two-stage probabilistic chase procedure that can be used to answer threshold deduplication query Q over KB.

Essentially, this result holds since the first stage of our chase procedure applies TGDs and dhTGDs (which are a special case of classical TGDs) in the order given by the stratification, and the second stage is simply an application of pEGDs over a fixed structure. Both stages involve carrying over probabilistic events, which are then used in conjunction with the probabilistic model to check which atoms surpass the threshold given in the query.

6 Conclusions and Future Work

In this paper, we have introduced the *adversarial deduplication problem* as a variant of the classical deduplication (or entity resolution) problem in which one cannot assume that the existence of multiple records corresponding to the same real-world entity is the result of innocent or casual errors such as typos, transliterations, or inconsistently-updated values, but rather the result of actors deliberately trying to avoid identification. As a motivating scenario we took malicious hacker forums and marketplaces on the dark web, where this sort of practice is commonplace and having tools at hand to address it would have a great impact on efforts such as early-warning of cyber attacks, understanding the social networks of buyers and sellers, and identification of leading vendors of specific types of products.

We proposed the use of probabilistic Datalog+/– as a way to tackle the problem of finding pairs of virtual objects for which we have reasons to believe that they correspond to the same real-world entity, where a specific kind of probabilistic TGDs (called deduplication hypothesis-generating dependencies) and EGDs can be applied in order to derive deduplication hypotheses. The main strength of this approach is that it allows to reason about unknown objects via value invention, which is essential in this setting—not having this capability renders traditional entity resolution tools ineffective. Another advantage of our approach is that dhTGDs can be automatically obtained via relational association rule mining algorithms, and corresponding EGDs can also be derived with additional domain knowledge. Finally, we developed a two-stage probabilistic chase procedure in which (dh)TGDs are first exhaustively applied, and then pEGDs are used to associate concrete values to null values generated in the first step. We showed that this modified chase procedure can be used to answer deduplication threshold queries, leveraging chase termination results from the existential rules and Datalog+/– literature.

Ongoing and future work involves investigating other kinds of deduplication queries, such as instance checking ("what is the probability that two given virtual objects correspond to the same real-world entity?"), all pairs threshold ("which pairs of virtual objects are suspected to correspond to the same real-world entity with probability at least p?"), and more general conjunctive queries. We are also interested in investigating the complexity of query answering algorithms for all such queries, and under which conditions they can be guaranteed to run in polynomial time. We plan to evaluate the performance of our approach on both synthetic and real-world data on malicious hacker forums and marketplaces on the dark web obtained via a collaboration with a cyber threat intelligence company. Finally, we would also like to explore the relationship between our formalism and other approaches to probabilistic modeling with unknown objects, such as [24]

Acknowledgments. This work was supported by funds provided by CONICET, Agencia Nacional de Promoción Científica y Tecnológica, Universidad Nacional del Sur, Argentina, by the EU H2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement 690974 for the project "MIREL: MIning and REasoning with Legal texts", and by the US Department of the Navy, Office of Naval Research, grant N00014-15-1-2742. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the Office of Naval Research.

References

1. Abiteboul, S., Hull, R., Vianu, V. (eds.): Foundations of Databases: The Logical Level. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1995)
2. Baget, J., Mugnier, M., Rudolph, S., Thomazo, M.: Walking the complexity lines for generalized guarded existential rules. In: Proce. IJCAI. pp. 712–717 (2011)
3. Bahmani, Z., Bertossi, L.E., Vasiloglou, N.: Erblox: Combining matching dependencies with machine learning for entity resolution. Int. J. Approx. Reasoning 83, 118–141 (2017)
4. Beeri, C., Vardi, M.Y.: A proof procedure for data dependencies. J. ACM 31(4), 718–741 (1984)
5. Bertossi, L.E., Kolahi, S., Lakshmanan, L.V.S.: Data cleaning and query answering with matching dependencies and matching functions. Theory Comput. Syst. 52(3), 441–482 (2013)
6. Bhattacharya, I., Getoor, L.: Collective entity resolution in relational data. ACM Trans. Knowl. Discov. Data 1(1) (2007)
7. Bhattacharya, I., Getoor, L.: Query-time entity resolution. J. Artif. Intell. Res. 30, 621–657 (2007)
8. Bleiholder, J., Naumann, F.: Data fusion. ACM Comput. Surv. 41(1), 1–41 (2009)
9. Calì, A., Gottlob, G., Kifer, M.: Taming the infinite chase: Query answering under expressive relational constraints. J. Artif. Intell. Res. 48, 115–174 (2013)

10. Calì, A., Gottlob, G., Pieris, A.: Towards more expressive ontology languages: The query answering problem. Artif. Intell. 193, 87–128 (2012)
11. CVE: Common vulnerabilities and exposures: The standard for information security vulnerability names. http://cve.mitre.org/ (2018)
12. Dehaspe, L., Toivonen, H.: Discovery of relational association rules. In: Relational data mining, pp. 189–212. Springer (2001)
13. Diaz-Valenzuela, I., Martin-Bautista, M.J., Vila, M.A., Campaña, J.R.: An automatic system for identifying authorities in digital libraries. Expert Syst. Appl. 40(10), 3994–4002 (2013)
14. Elmagarmid, A.K., Ipeirotis, P.G., Verykios, V.S.: Duplicate record detection: A survey. IEEE T. Knowl. Data En. 19(1), 1–16 (2007)
15. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: semantics and query answering. Theor. Comput. Sci. 336(1), 89–124 (2005)
16. Fan, W.: Dependencies revisited for improving data quality. In: Proc. ACM PODS. pp. 159–170 (2008)
17. Fan, W., Jia, X., Li, J., Ma, S.: Reasoning about record matching rules. PVLDB 2(1), 407–418 (2009)
18. Getoor, L., Machanavajjhala, A.: Entity resolution: Theory, practice and open challenges. PVLDB 5(12), 2018–2019 (2012)
19. Gottlob, G., Lukasiewicz, T., Martinez, M.V., Simari, G.I.: Query answering under probabilistic uncertainty in Datalog+/– ontologies. Ann. Math. Artif. Intell. 69(1), 37–72 (2013)
20. Gottlob, G., Rudolph, S., Simkus, M.: Expressiveness of guarded existential rule languages. In: Proc. ACM PODS. pp. 27–38 (2014)
21. Leone, N., Manna, M., Terracina, G., Veltri, P.: Efficiently computable datalog∃ programs. In: Proc. KR (2012)
22. Lukasiewicz, T., Cali, A., Gottlob, G.: A general Datalog-based framework for tractable query answering over ontologies. J. Web Semant. 14(0) (2012)
23. Maier, D., Mendelzon, A.O., Sagiv, Y.: Testing implications of data dependencies. ACM T. Database Syst. 4(4), 455–469 (1979)
24. Milch, B., Marthi, B., Russell, S.J., Sontag, D., Ong, D.L., Kolobov, A.: BLOG: probabilistic models with unknown objects. In: Proc. IJCAI. pp. 1352–1359 (2005)
25. NIST: National Vulnerability Database. https://nvd.nist.gov/ (2018)
26. Nunes, E., Diab, A., Gunn, A.T., Marin, E., Mishra, V., Paliath, V., Robertson, J., Shakarian, J., Thart, A., Shakarian, P.: Darknet and deepnet mining for proactive cybersecurity threat intelligence. In: Proc. ISI. pp. 7–12 (2016)
27. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann Publishers Inc. (1988)
28. Richardson, M., Domingos, P.: Markov logic networks. Mach. Learn. 62(1-2), 107–136 (2006)
29. Simari, G.I., Molinaro, C., Martinez, M.V., Lukasiewicz, T., Predoiu, L.: Ontology-Based Data Access Leveraging Subjective Reports. Springer Briefs in Computer Science, Springer (2017)
30. Whang, S.E., Menestrina, D., Koutrika, G., Theobald, M., Garcia-Molina, H.: Entity resolution with iterative blocking. In: Proc. SIGMOD. Stanford (2009)