On Data Dependencies in Dataspaces

Shaoxu Song #1, Lei Chen #2, Philip S. Yu *3

#The Hong Kong University of Science and Technology, Hong Kong
1sshaoxu@cse.ust.hk 2leichen@cse.ust.hk

*University of Illinois at Chicago, Chicago, IL, USA
3psyu@cs.uic.edu

***Abstract*—To study data dependencies over heterogeneous data in dataspaces, we define a general dependency form, namely *comparable dependencies* (CDs), which specifies constraints on comparable attributes. It covers the semantics of a broad class of dependencies in databases, including *functional dependencies* (FDs), *metric functional dependencies* (MFDs), and *matching dependencies* (MDs). As we illustrated, comparable dependencies are useful in real practice of dataspaces, e.g., semantic query optimization. Due to the heterogeneous data in dataspaces, the first question, known as the validation problem, is to determine whether a dependency (almost) holds in a data instance. Unfortunately, as we proved, the validation problem with certain error or confidence guarantee is generally hard. In fact, the confidence validation problem is also NP-hard to approximate to within any constant factor. Nevertheless, we develop several approaches for efficient approximation computation, including greedy and randomized approaches with an approximation bound on the maximum number of violations that an object may introduce. Finally, through an extensive experimental evaluation on real data, we verify the superiority of our methods.**

I. Introduction

The importance of dataspace systems has already been recognized and emphasized in handling heterogeneous data [1], [2], [3], [4], [5]. Generally, dataspaces consider three levels of elements, *object, attribute, value*, in the form of $object : \{(attribute : value)\}$. We illustrate a sample dataspace of several product objects with certain attribute value pairs in Example I.1.

Example I.1. *We consider a dataspace with 3 objects,*

$t_1 : \{(\mathsf{name} : \mathsf{iPod}), (\mathsf{color} : \mathsf{red}), (\mathsf{manu} : \mathsf{Apple\ Inc.}), (\mathsf{tel} : 567),$
$(\mathsf{addr} : \mathsf{InfiniteLoop, CA}), (\mathsf{website} : \mathsf{itunes.com})\};$

$t_2 : \{(\mathsf{name} : \mathsf{iPod}), (\mathsf{color} : \mathsf{cardinal}), (\mathsf{prod} : \mathsf{Apple}), (\mathsf{tel} : 123),$
$(\mathsf{post} : \mathsf{InfiniteLoop, Cupert}), (\mathsf{website} : \mathsf{apple.com})\};$

$t_3 : \{(\mathsf{name} : \mathsf{iPad}), (\mathsf{color} : \mathsf{white}), (\mathsf{manu} : \mathsf{Apple\ Inc.}),$
$(\mathsf{post} : \mathsf{InfiniteLoop}), (\mathsf{website} : \mathsf{apple.com}), (\mathsf{phn} : 567)\},$

where manu *denotes an attribute of manufacturer,* prod *is producer, and* addr *denotes address.*

The comparable correspondences between values (e.g., Apple vs. Apple Inc) on comparable attributes (e.g., manu vs. prod) denote the synonym relationships between elements from heterogeneous sources. They are often recognized incrementally in a pay-as-you-go style [3], i.e., gradually identified according to users' feedback when necessary. For instance, a comparison operator '$\mathsf{manu} \approx_{\leq 5} \mathsf{prod}$' states that any two respective values of manu and prod are said comparable, e.g., Apple Inc and Apple, if their edit distance is ≤ 5. Such comparable correspondence on metric distance of values is often obtained by a *metric operator* [6]. Moreover, the matching correspondence [3] can also be identified between two elements which denote the same entity in real world, e.g., red and cardinal are said *matched* as comparable color. The matching correspondence is usually confirmed by a *matching operator*, e.g., via update with dynamic semantics [7] or users' feedback [3].

Data dependencies have already shown their importance in various data-oriented practice [8], such as optimizing query evaluation [9], capturing data inconsistency [10], removing data duplicates [7], etc. It is promising to study data dependencies for the heterogeneous data in dataspaces as well. Unfortunately, few works have been drawn to address such data dependencies. Although, Wang et al., [11] extends functional dependencies (FDs) with probability for data integration systems, namely *probabilistic functional dependencies* (pFDs), this extension of FDs is still declared based on the equality of values and not directly applicable in dataspaces. As mentioned before, data values in dataspaces are highly heterogeneous with various comparable correspondences instead of precise equality.

In this paper, to adapt data dependencies to dataspaces, we introduce a general *comparison function* to specify the comparable correspondences on attributes with respect to various comparison operators.

Example: Intuitively, the comparable correspondence may occur either between the same attribute (e.g., $t_1[\mathsf{manu}]$ vs. $t_3[\mathsf{manu}]$) or between comparable attributes (e.g., $t_1[\mathsf{manu}]$ vs. $t_2[\mathsf{prod}]$). A comparison function on two attributes $(\mathsf{manu}, \mathsf{prod})$ w.r.t. metric operators in Example I.1 can be

$$\theta(\mathsf{manu}, \mathsf{prod}) : [\mathsf{manu} \approx_{\leq 5} \mathsf{manu}, \mathsf{manu} \approx_{\leq 5} \mathsf{prod}, \mathsf{prod} \approx_{\leq 5} \mathsf{prod}].$$

Two objects are said comparable on $(\mathsf{manu}, \mathsf{prod})$ if at least one of these three comparison operators in $\theta(\mathsf{manu}, \mathsf{prod})$ is evaluated to be true. For example, t_1, t_2 are comparable on $(\mathsf{manu}, \mathsf{prod})$, since edit distance of $(t_1[\mathsf{manu}], t_2[\mathsf{prod}])$ is ≤ 5. Similarly, t_1, t_3 are also comparable on $(\mathsf{manu}, \mathsf{prod})$, where $(t_1[\mathsf{manu}], t_3[\mathsf{manu}])$ satisfy '$\mathsf{manu} \approx_{\leq 5} \mathsf{manu}$'. Let

$$\theta(\mathsf{addr}, \mathsf{post}) : [\mathsf{addr} \approx_{\leq 9} \mathsf{addr}, \mathsf{addr} \approx_{\leq 9} \mathsf{post}, \mathsf{post} \approx_{\leq 9} \mathsf{post}]$$

be another comparison function. A general form of dependencies is then defined on such comparison functions, namely *comparable dependencies* (CDs), e.g.,

$$\varphi_1 : \theta(\mathsf{manu}, \mathsf{prod}) \rightarrow \theta(\mathsf{addr}, \mathsf{post}).$$

It states that if the manu or prod values of two objects are comparable, then their corresponding addr or post values should also be comparable. As data dependencies have been found useful in various data-oriented applications [8], comparable dependencies are also promising for dataspace applications.

Applications: In semantic query optimization [12], [9], [13], a conjunctive query can be rewritten by using part of predicates according to data dependencies. Such optimization can also be introduced to queries in dataspaces on comparable attributes. For example, we consider a query object with (manu : Apple) and (post : InfiniteLoop, CA). The query evaluation [14] searches not only in the manu, post attributes specified in the query, but also in the comparable attributes prod, addr according to the comparison functions $\theta(\mathsf{manu}, \mathsf{prod})$ and $\theta(\mathsf{addr}, \mathsf{post})$, respectively. Recall the semantics of the above dependency φ_1. If (manu, prod) of the query object and a data object are found comparable, then the data object can be directly returned as answer without evaluation on post, addr since their corresponding (post, addr) values must be comparable as well. Consequently, the query efficiency is improved.

To give another example, data dependencies are often employed to handle violations in data, i.e., those objects not satisfying data dependencies. Methods are proposed to remove such violations by repairing [8]. Similarly, we can also utilize the comparable dependencies to detect and handle violations in dataspaces. For instance, one may detect a minimum set of tuples as violations which do not satisfy the given CDs. Moreover, dataspaces often collect data from various sources without permission to write (repair). Thus, it is particularly important to study consistent query answering [15], which can return objects that have no violations without updating original data in dataspaces.

Challenges: To our best knowledge, this is the first work on adapting data dependencies to the heterogeneous data in dataspaces. Unfortunately, it is highly non-trivial to study data dependencies in dataspaces, with the consideration of comparable correspondence. Due to the extremely high heterogeneity, data dependencies may "almost/approximately" hold in dataspaces. As illustrated below, it is already hard to determine whether a dependency approximately hold in a dataspace, i.e., the validation problem. Moreover, since comparable correspondences are often identified in an incremental style, namely pay-as-you-go, an incremental discovery of data dependencies is urgent and more challenging, which is not considered in the previous dependency discovery problem.

Contributions: Our main contributions in this paper are summarized as follows.

i) We formalize the notations of data dependencies in dataspaces. Our *comparable dependencies* (CDs) on comparison functions cover the semantics of a broad class of well known dependencies in databases, such as functional dependencies, metric functional dependencies [16] and matching dependencies [8]. We introduce the widely used error and confidence measures to evaluate how a dependency approximately holds in a dataspace.

ii) We characterize the validation problem of dependencies in dataspaces. As we prove, the computation of confidence or error measures for approximate dependencies with general comparison functions is NP-hard. The corresponding decision version, i.e., the validation problem, is NP-complete. In fact, even the special case of aligned attributes, i.e., with $\theta(A_i, A_i)$ only, is still NP-complete.

iii) We develop novel approximation methods to compute confidence and error measures. According to our theoretical analysis, the error measure computation can be approximated in polynomial time with a constant factor, while the confidence has no constant-factor approximation solution unless P=NP. We develop the greedy approaches with an approximation ratio on the bound of violations that an object may introduce. Moreover, the randomized approaches are further developed to improve the efficiency, where the approximation bound with an additive error is obtained with a high probability.

iv) We introduce a pay-as-you-go approach for discovering dependencies in a given dataspace. Note that comparable correspondences are often identified in an incremental style, namely pay-as-you-go. Thereby, we investigate a framework which incrementally discovers data dependencies with respect to the newly identified comparison function.

v) Finally, we report an extensive experimental evaluation of the proposed approaches on real data sets. Both the effectiveness and efficiency of the proposed approximation computation techniques are illustrated. The discovery performance is also evaluated on real data sets.

The remainder of this paper is organized as follows. First, we discuss the related work in Section II. Then, in Section III, we study the foundations of comparable dependencies in dataspaces. Section IV introduces the validation problem of approximate dependencies. In Section V, we develop efficient approaches for computing error and confidence measures. Section VI studies the pay-as-you-go discovery of comparable dependencies in dataspaces. In Section VII, we report our extensive experimental results. Finally, Section VIII concludes this paper.

II. Related Work

Although data dependencies have been well studied in databases, few works are drawn over heterogeneous data, especially in dataspaces with comparable correspondences. Table I lists the most typical related works, compared with our CDs.

Due to the data heterogeneity, data dependencies might not exactly hold in the entire database of all tuples. Therefore, *conditional functional dependencies* (CFDs), as an extension of traditional FDs with conditions, are first proposed in [10] for data cleaning. The basic idea of CFDs is making the FDs, originally hold for the whole table, valid only for a set of tuples specified by the conditions. However, the equality function is

TABLE I
RELATED WORK

Dependencies	Operators	Comparable correspondences	Measures	Validation
pFDs [11]	Equality operator	Cannot address	Probability	PTIME
CFDs [10]	Equality operator	Cannot address	Confidence and condition support	PTIME
MFDs [16]	Equality and metric operators	Cannot address	Not studied	Not studied
MDs [8]	Metric and matching operators	Cannot address	Not studied	Not studied
Our CDs	All the above operators	Address	Error or confidence	NP-complete

still considered in CFDs, which cannot address the various information formats of data from different sources, especially in dataspaces. Note that the error (confidence) measure is also used in evaluating and discovering CFDs [17], [18]. Moreover, since CFDs are declared over a subset of tuples specified by conditions, a support measure is further introduced to report the number of tuples agreeing the given condition.

Wang et al., [11] extends functional dependencies with probability for data integration systems, namely *probabilistic functional dependencies* (pFDs), which is similar to the concepts of approximate FDs [19] and soft FDs [20]. Given a mediated schema and mappings from each source to the mediated schema, the probability of an FD in each data source is merged together as a global measure. However, this extension of traditional FDs is still on equal values and not directly applicable to dataspaces, where data values are highly heterogeneous with various comparable correspondences instead of precise equality. In particular, dataspace systems often provide services without investigating a mediated schema as traditional data integration systems. Thus, data dependencies incorporating with comparison functions are necessary in dataspaces.

Koudas et al. [16] study *metric functional dependencies* (MFDs) with metric operator on attribute A when given the equality on X, where X and A are attributes in relation schema R. An MFD has the form $X \rightarrow {}^{\lambda}A$, where $\lambda \geq 0$ is a threshold of metric distance on A. For example, an MFD can be $\texttt{name} \rightarrow^{9} \texttt{addr}$. In our work, we adapt such dependencies with metrics to dataspaces by introducing comparison functions. Note that techniques in [16] only verify whether or not an MFD exactly holds, i.e., exact MFDs, while the validation problem is not studied to determine whether a dependency almost/approximately holds with certain error/confidence guarantee, i.e., approximate MFDs. As we investigated in special cases (Section IV), the validation problem of approximate MFDs in databases is NP-complete, which is not addressed in previous works.

Fan [8] proposes *matching dependencies* (MDs) for specifying matching rules for object identification. An MD across two relations has a form $\bigwedge[A_i \approx_i B_i] \rightarrow [Y_1 \rightleftharpoons Y_2]$, where A_i and Y_1 are attributes in relation schema R_1, B_i and Y_2 are attributes in R_2, and $\approx_i, \rightleftharpoons$ denote the corresponding metric/matching operators on attributes of (A_i, B_i) and (Y_1, Y_2), respectively. It states that for any two tuples from the instance of relations R_1 and R_2, respectively, if they are similar ($\approx$) on attributes in the left-hand-side, then their right-hand-side Y_1, Y_2 values should be matched ($\rightleftharpoons$). Reasoning mechanism for deducing MDs from a set of given MDs is studied in [7]. The MDs and their reasoning techniques can improve both the quality and efficiency of object identification methods. As one type of dependencies we considered in dataspaces, MDs are naturally applicable to identify duplicates in dataspaces as well. Again, the validation problem of MDs is not considered in previous works.

III. FOUNDATIONS

In this section, we address the fundamental issues of adapting data dependencies to the heterogeneous data in dataspaces. Table II lists the frequently used notations.

TABLE II
NOTATIONS

Symbol	Description
$\mathcal{S}$	a dataspace
$A_i \leftrightarrow_{ij} A_j$	a comparison operator of attribute A_i, A_j
$\theta(A_i, A_j)$	a general comparison function on attribute A_i, A_j
φ	a dependency with general comparison functions
g	error measure of a dependency
c	confidence measure of a dependency
η	requirement of a measure
ρ	approximation ratio
δ	approximation probability guarantee
ϵ	approximation additive error
Δ	a bound of violations of an object

A. Comparable Dependencies

Let $\leftrightarrow_{ij}$ denote a *comparison operator* between two attributes A_i, A_j in a dataspace $\mathcal{S}$, which is a generic operator that can have either one of the following semantics.

- $\leftrightarrow_{ij}$ can be an *equality operator* $A_i = A_j$, which is considered in traditional functional dependencies. Let a_i, a_j be the values of A_i, A_j, respectively. The comparison operator $=$ evaluates to $\texttt{true}$ if $a_i = a_j$, i.e., identical.
- $\leftrightarrow_{ij}$ can also be a *metric operator* $A_i \approx_\lambda A_j$, which is raised in metric functional dependencies [16]. Let d_{ij} be a distance metric[1] defined on the domains of two comparable attributes A_i, A_j. The metric operator $\approx_\lambda$

[1]For example, the absolute value of difference on numerical values, or edit distance on string values (see [6] for a survey).

evaluates to true if $d_{ij}(a_i, a_j) \leq \lambda$, i.e., the metric distance is less than a threshold λ.
- $\leftrightarrow_{ij}$ could be a matching operator as well, $A_i \rightleftharpoons A_j$, which is studied in matching dependencies [8]. The matching operator $\rightleftharpoons$ evaluates to true if a_i and a_j are identified as matched, i.e., make them identical via dynamic semantics [8] or by users' feedback [3].

The comparison operator $\leftrightarrow_{ij}$ can have any one of the above listed semantics. For each pair of attributes A_i, A_j in $\mathcal{S}$, we consider **one** comparison operator on them. There may also have a comparison operator associated between the same attribute A_i, i.e., $\leftrightarrow_{ii}$.

Comparison Function: A general *comparison function*

$$\theta(A_i, A_j) : [A_i \leftrightarrow_{ii} A_i, A_i \leftrightarrow_{ij} A_j, A_j \leftrightarrow_{jj} A_j]$$

specifies a constraint on comparable correspondence of two values from attribute A_i or A_j, according to their corresponding comparison operators $\leftrightarrow_{ii}, \leftrightarrow_{ij}$ or $\leftrightarrow_{jj}$.

Definition III.1. *Given any two objects t_1, t_2 in the dataspace, we say that t_1, t_2* agree *on a comparison function, denoted by*

$$(t_1, t_2) \asymp \theta(A_i, A_j),$$

if at least one pair of $(t_1[A_i], t_2[A_i]), (t_1[A_i], t_2[A_j]), (t_1[A_j], t_2[A_i])$ or $(t_1[A_j], t_2[A_j])$ agrees on the corresponding comparison operator specified by $\theta(A_i, A_j)$, i.e.,

$$(t_1[A_i] \leftrightarrow_{ii} t_2[A_i]) \vee (t_1[A_i] \leftrightarrow_{ij} t_2[A_j]) \vee$$

$$(t_2[A_i] \leftrightarrow_{ij} t_1[A_j]) \vee (t_1[A_j] \leftrightarrow_{jj} t_2[A_j]) = \mathsf{true};$$

otherwise, disagree, *denoted by $(t_1, t_2) \not\asymp \theta(A_i, A_j)$*

For instance, we have $(t_1, t_2) \asymp \theta(\mathsf{manu}, \mathsf{prod})$ in Example I.1, since the edit distance between $t_1[\mathsf{manu}]$ and $t_2[\mathsf{prod}]$ is ≤ 5. In a special case, a comparison function with aligned attributes A_i is $\theta(A_i, A_i) : [A_i \leftrightarrow_{ii} A_i]$.

It is worth noting that a sophisticated comparison operator may interact with more than two attributes, e.g., among $(\mathsf{addr}, \mathsf{street}, \mathsf{city})$. Since the comparison operator is not the focus of this study, we will leave such comparison operators as our future work.

Comparable Dependency: A *comparable dependency* (CD) φ with general comparison functions over a dataspace $\mathcal{S}$ is in the form[2] of

$$\varphi : \quad \bigwedge \theta(A_i, A_j) \rightarrow \theta(B_1, B_2),$$

where $\theta(A_i, A_j)$ and $\theta(B_1, B_2)$ are comparison functions in the dataspace $\mathcal{S}$. We denote $\bigwedge \theta(A_i, A_j)$ and $\theta(B_1, B_2)$ by the LHS and RHS of φ, respectively.

Definition III.2. *Given any two objects t_1, t_2 in the dataspace $\mathcal{S}$, we say that t_1, t_2* satisfy *a dependency φ, denoted by $(t_1, t_2) \vDash \varphi$, if $(t_1, t_2) \asymp \text{LHS}(\varphi)$ implies $(t_1, t_2) \asymp \theta(B_1, B_2)$.*

[2]In this study, we focus on dependencies in the standard form with only one function in the right-hand-side. Dependencies with multiple comparison functions in RHS can be naturally inferred according to the reflexivity and augmentation rules.

That is, if t_1, t_2 agree on all the comparison functions $\theta(A_i, A_j)$ in the left-hand-side of the dependency φ, then they must also agree on the right-hand-side comparison function $\theta(B_1, B_2)$.

Definition III.3. *Given a dataspace $\mathcal{S}$ and a dependency φ, we say that the dataspace $\mathcal{S}$* satisfies *φ or the dependency φ* holds *in $\mathcal{S}$, denoted by $\mathcal{S} \vDash \varphi$, if any two objects t_1, t_2 from $\mathcal{S}$ always have $(t_1, t_2) \vDash \varphi$, i.e., any two objects in $\mathcal{S}$ satisfy φ.*

Example III.1. *Consider the dataspace $\mathcal{S}$ as illustrated in Example I.1. A comparable dependency can be*

$$\varphi_2 : [\mathsf{name} = \mathsf{name}] \rightarrow \theta(\mathsf{manu}, \mathsf{prod}),$$

which states that if two products have the same name*, then their* (manu, prod) *should be comparable.*

To give another example, we consider

$$\theta(\mathsf{website}, \mathsf{website}) : [\mathsf{website} \rightleftharpoons \mathsf{website}].$$

The following comparable dependency

$$\varphi_3 : \theta(\mathsf{manu}, \mathsf{prod}) \wedge \theta(\mathsf{addr}, \mathsf{post}) \rightarrow \theta(\mathsf{website}, \mathsf{website}),$$

states that if two product objects have comparable (manu, prod) *and comparable* (addr, post)*, then their* website *should be matched, e.g.,* itunes.com *and* apple.com *are matched* URL*s of the same web site.*

It is notable that comparable dependencies are different from the concept of tail in dataspaces [5], [21]. A tail is denoted as $Q_L[.C_L] \rightarrow Q_R[.C_R]$, which means that the query on the left $Q_L[.C_L]$ includes the query on the right $Q_R[.C_R]$. In other words, whenever we query for $Q_L[.C_L]$ we should also query for $Q_R[.C_R]$. Thereby, it can naturally be extended to a bidirectional trail such as $Q_L[.C_L] \leftrightarrow Q_R[.C_R]$. Instead of the comparison relationship between query components $Q_L[.C_L]$ and $Q_R[.C_R]$, we study the dependencies upon different comparison relationships.

B. Approximate Dependencies

Due to the extremely high heterogeneity, data dependencies might not exactly hold in a given dataspace, i.e., the dataspace approximately satisfies the dependencies. Given a pair of objects t_1, t_2 from dataspace $\mathcal{S}$, we say that (t_1, t_2) *violates* a dependency φ, denoted by $(t_1, t_2) \nvDash \varphi$, if $(t_1, t_2) \asymp \text{LHS}(\varphi)$ but $(t_1, t_2) \not\asymp \text{RHS}(\varphi)$.

For instance, let

$$\theta(\mathsf{tel}, \mathsf{phn}) : [\mathsf{tel} = \mathsf{tel}, \mathsf{tel} = \mathsf{phn}, \mathsf{phn} = \mathsf{phn}]$$

be a comparison function in Example I.1. We consider a dependency

$$\varphi_4 : \theta(\mathsf{manu}, \mathsf{prod}) \rightarrow \theta(\mathsf{tel}, \mathsf{phn}).$$

We have $(t_1, t_3) \vDash \varphi_4$ but $(t_1, t_2) \nvDash \varphi_4$, since t_1, t_2 agree on the left-hand-side $\theta(\mathsf{manu}, \mathsf{prod})$ of φ_4 but have different tel. Such data dependencies that "almost" or "approximately" hold in the data with some violations are called approximate dependencies [19], [22].

Error Measure: To evaluate how a dependency almost/approximately holds in a data instance, g_3 error measure [19] is widely used [22], [23]. We can also adopt this measure to evaluate a dependency φ over a dataspace $\mathcal{S}$.

Definition III.4. *The g_3 measure evaluates the minimum number of objects that have to be removed from dataspace $\mathcal{S}$ for the dependency φ to hold, i.e.,*

$$g_3(\varphi, \mathcal{S}) = \frac{|\mathcal{S}| - \max\{|T| \mid T \subseteq \mathcal{S}, T \vDash \varphi\}}{|\mathcal{S}|},$$

where T is a subset of objects in $\mathcal{S}$ that do not violate φ.

We call $V = \mathcal{S} \setminus T$ a candidate *violation set* such that all the objects in T satisfy the given dependency φ.

For example, $\{t_2\}$ in Example I.1 is a minimum violation set w.r.t. the above φ_4, such that all the remaining objects $\{t_1, t_3\}$ satisfy φ_4. Thereby, we have $g_3 = 1/3$.

Confidence Measure: Instead of measuring violations, a confidence [17], [24] defines the proportion of tuples after removing minimum tuples (objects in dataspace) of violations w.r.t. φ.

Definition III.5. *The* confidence *measure evaluates the maximum number of objects T from the dataspace $\mathcal{S}$ such that the dependency φ holds in T,*

$$\mathsf{conf}(\varphi, \mathcal{S}) = \frac{\max\{|T| \mid T \subseteq \mathcal{S}, T \vDash \varphi\}}{|\mathcal{S}|},$$

where T is a subset of objects in $\mathcal{S}$ that do not violate φ.

As claimed in [11], this confidence measure is close to the concept of probability of FDs, which also considers the maximum number of tuples (objects) that follow the dependency. We call the above T a candidate *keeping set* such that all the objects in $T \subseteq \mathcal{S}$ satisfy the given dependency φ, i.e., non-violation.

For instance, $\{t_1, t_3\}$ in Example I.1 is a maximum keeping set w.r.t. the above φ_4, having $\mathsf{conf} = 2/3$.

Note that the g_3 measure denotes the minimum number of objects removed, while the confidence measure reports the maximum number of objects reserved, which are essentially equivalent. When $g_3 = 0$ or equivalently $\mathsf{conf} = 1$, it is the case of exact dependency.

IV. Validation Problem

Given a dataspace and a dependency, the validation problem is to determine whether or not the dependency (approximately) holds in the dataspace. Recall that confidence and error measures tell how a dependency approximately holds in a data set. Thereby, we can formalize the validation problem as follows.

Problem IV.1. *Given a dataspace $\mathcal{S}$ with n objects, the* validation problem *is to decide whether or not the error/confidence measure of a dependency φ over $\mathcal{S}$ satisfies the measure requirement η. Specifically, the error validation problem is to determine whether $g_3(\varphi, \mathcal{S}) \leq \eta_e$, and the confidence validation problem is to determine whether* $\mathsf{conf}(\varphi, \mathcal{S}) \geq \eta_c$.

The Hardness: Essentially, given a dataspace and a dependency, we are required to compute the error and confidence of the dependency in the dataspace. For FDs and their extensions in traditional databases, polynomial time algorithms can be developed to efficiently compute these measures [23], [24]. Unfortunately, in the scenario of dataspaces with general comparison functions, the problem of computing error/confidence is highly non-trivial. Intuitively, the transitivity is no longer valid on a general comparison function, i.e., from $(t_1, t_2) \asymp \theta(A_i, A_j)$ and $(t_2, t_3) \asymp \theta(A_i, A_j)$, it does not necessarily follow that $(t_1, t_3) \asymp \theta(A_i, A_j)$. See the following Example IV.1 for instance. The efficient validation computation based on disjoint grouping [23] cannot be applied to this case with general functions. In fact, we can prove the hardness of validation problem.

Theorem IV.1. *Both the error and confidence validation problems with general comparison functions are* NP*-complete.*

Proof sketch: We show that the VERTEX COVER problem, which is one of Karp's 21 NP-complete problems [25], is reducible to the error validation problem. Moreover, the CLIQUE problem is reducible to the confidence validation problem. ■

The above conclusion is not only valid for CDs in dataspaces but also valid for MDs in databases. Intuitively, the transitivity is not valid either on the metric operators used in MDs. In fact, our proof of Theorem IV.1 is sufficient to show that the validation of MDs is also NP-complete.

Special Case of Aligned Attributes: An interesting special case is to consider comparison functions on aligned attributes, i.e., $\theta(A_i, A_i)$. In other words, it is a case without comparable attribute pairs, which is similar to the traditional database scenario. Unfortunately, we cannot assume the transitivity on such case either.

Example IV.1. *We consider a function on aligned attributes $\theta(A_1, A_1) : [A_1 \approx_{\leq 1} A_1]$ with edit distance as metric d. Let*

$$\begin{aligned} t_1 &: \{(A_1 : \mathit{abc}), \dots\}; \\ t_2 &: \{(A_1 : \mathit{abcd}), \dots\}; \\ t_3 &: \{(A_1 : \mathit{abcde}), \dots\}. \end{aligned}$$

We have edit distance

$$d(t_1[A_1], t_2[A_1]) = 1 \leq 1, \text{ and}$$

$$d(t_2[A_1], t_3[A_1]) = 1 \leq 1, \text{ but}$$

$$d(t_1[A_1], t_3[A_1]) = 2 > 1,$$

that is, $(t_1, t_2) \asymp \theta(A_1, A_1)$, $(t_2, t_3) \asymp \theta(A_1, A_1)$ but $(t_1, t_3) \not\asymp \theta(A_1, A_1)$.

Therefore, it is not surprising that the validation problem is still hard.

Theorem IV.2. *The error/confidence validation problem with aligned attributes in general comparison functions is still* NP*-complete.*

Proof sketch: Similar to the proof of Theorem IV.1, a reduction from VERTEX COVER/CLIQUE problem can be constructed. In particular, to prove that the confidence validation problem with aligned attributes is NP-hard, we build a polynomial-time reduction from the CLIQUE problem. The proof of error validation problem follows a similar reduction from the VERTEX COVER problem as shown in the proof of Theorem IV.1. ■

Moreover, our proof of Theorem IV.2 is also sufficient to find that the error/confidence validation problem for MFDs in databases is also NP-complete. As a special case, our approximation computation techniques proposed below can be naturally applied to computing error/confidence of MFDs in databases as well.

Tractable Special Case: Finally, another special case with both aligned attributes and equality function, i.e., $[A_i = A_i]$, is exactly the case of FDs with equality in traditional databases. As mentioned, such a special case has been well investigated with efficient algorithms [23].

V. Approximation Computation

Motivated by the above hardness analysis of the validation problem, in this section, we study efficient approaches to approximately compute error and confidence measures.

Problem V.1. *Given a dataspace $\mathcal{S}$ and a dependency φ, the measure estimation problem is to compute an approximate error/confidence measure of φ over $\mathcal{S}$ such that the approximate measure has a relative performance guarantee compared with exact measure, e.g., $\hat{g}/g \leq \rho$ where $\hat{g}$ is an estimation of exact error measure g and ρ is approximation ratio.*

An even more relaxed version is to get the approximate measure with a high probability, e.g., $\Pr[\hat{g} \leq \rho g + \epsilon] \geq \delta$, *where ϵ is an additive error and δ is a probability guarantee.*

In Section V-A, we give greedy algorithms for computing approximate error and confidence with a relative performance guarantee ρ. Since greedy algorithms still have to consider all the objects in a dataspace, we also develop randomized algorithms in Section V-B, which estimate error and confidence measures upon only a small subset of objects, and are still guaranteed by certain approximation bound with a high probability δ.

A. Greedy Approach

Preliminary: Given a comparison function $\theta(A_i, A_j)$, we can obtain a set of object pairs which agree on the function, i.e., $\{(t_i, t_j) \in \mathcal{S} \mid (t_i, t_j) \asymp \theta(A_i, A_j)\}$. By an intersection of the sets of object pairs corresponding to the functions in LHS(φ), we obtain those object pairs agreeing on LHS(φ), say L. Similarly, let H be object pairs agreeing on both LHS(φ) and RHS(φ). Then, we have the set of object pairs which violates φ, i.e., $L \backslash H$. Based on these object pairs of violations, the error and confidence measures can be computed.

Approximate Error: We first present the approximation computation of g_3 error. According to the proof of Theorem IV.1, computing a minimum vertex cover of a graph, which corresponds to objects in dataspace $\mathcal{S}$ w.r.t. violations of a dependency φ, will yield a minimum violation set for g_3 measure computation. Although the problem is NP-hard, an efficient greedy algorithm with factor-2 approximation bound is known for the MINIMUM VERTEX COVER problem [26].

Specifically, we greedily count both objects when a violation to the dependency occurs, and move them to the violation set. The procedure terminates if no violation exists in the dataspace, and we report the proportion of objects in violation set as the estimated error measure. The pseudo-code of greedy computation is given in Algorithm 1. Let n be the number of objects in $\mathcal{S}$. Algorithm 1 returns an approximate error $\hat{g}$ with relative performance bounded by factor-2 as it is used for MINIMUM VERTEX COVER [26]. It is implemented in linear time w.r.t. the number of objects pairs that are associated by comparison functions. Considering possible object pairs in a dataspace $\mathcal{S}$ with n objects, we have the algorithm complexity $O(n^2)$.

Algorithm 1 Greedy ERROR($\varphi, \mathcal{S}$)

```
Input: A dependency $\varphi$ over a dataspace $\mathcal{S}$
Output: An estimated error measure $\hat{g}$
1: $V := \emptyset$
2: $L := \{(t_i, t_j) \in \mathcal{S} \mid (t_i, t_j) \asymp \text{LHS}(\varphi)\}$
3: for each pair $(t_i, t_j) \in L$ do
4:     if $(t_i, t_j) \not\asymp \text{RHS}(\varphi)$ and $t_i \notin V$ and $t_j \notin V$ then
5:         $V := V \cup \{t_i, t_j\}$
6: return $|V|/|\mathcal{S}|$
```

Proposition V.1. *The greedy Algorithm 1 outputs an estimate $\hat{g}$ with a bound $g \leq \hat{g} \leq 2g$ compared with the exact error measure g. The complexity is $O(n^2)$.*

Although slightly better yet more complicated approximations can be achieved, e.g., with an approximation factor of $2 - \Theta(\frac{1}{\sqrt{\log n}})$ in [27], we do not consider them here. In fact, due to the hardness of approximating minimum vertex cover problem [28], it is not surprising to have:

Theorem V.1. *The g_3 error is NP-hard to approximate to within any factor $10\sqrt{5} - 21 \approx 1.36067$.*

Proof sketch: We show that the reduction from the VERTEX COVER problem in the proof of Theorem IV.1 is a *gap-preserving* reduction. Since MINIMUM VERTEX COVER cannot be approximated within a factor of $\alpha = 10\sqrt{5} - 21 \approx 1.36067$ unless P=NP [28], the conclusion is proved. ■

Approximate Confidence: Unfortunately, although computing the g_3 error is equivalent to finding the confidence, these two problems are not equivalent in an approximation-preserving way.

Theorem V.2. *The confidence has no constant-factor approximation unless* P=NP.

Proof sketch: To show the hardness of confidence approximation, we give a gap-preserving reduction from the CLIQUE problem. As studied in [29], the MAXIMUM CLIQUE problem is NP-hard to approximate to within any constant factor α. ■

Despite the hardness of approximation with respect to a constant factor, a heuristic greedy algorithm is known for the maximum independent set problem with performance ratio on the maximum degree in a graph [30]. We thus greedily compute approximate confidence as follows.

Specifically, we can iteratively select an object which has the minimum violations with others, move it to a keeping set, and delete all the objects having violations with this object. The procedure terminates when no object is left and reports the proportion of the number of objects in the keeping set as the estimated confidence measure. The pseudo-code with details is given in Algorithm 2. Let Δ be the maximum number of violations that an object may introduce in the dataspace. That is, an object violates with no more than Δ objects. It is often the case according to our observation in Section VII that the real data are extremely sparse. Algorithm 2 computes an approximate confidence $\hat{c}$. The operator $\arg\min$ in line 5 stands for the argument of the minimum, that is to say, the objects u in U for which the given expression has the minimum value. It can be done in constant time by amortizing objects $u \in U$ into an integer domain of $[1, \Delta]$. Considering possible pairs of n objects in a dataspace $\mathcal{S}$, we have the algorithm complexity $O(n^2)$. Note that a clique in a graph equals to an independent set in the corresponding graph's complement. Thus, the greedy algorithm gives a $(\Delta + 2)/3$ approximation ratio as it is used for MAXIMUM INDEPENDENT SET [30].

Algorithm 2 Greedy CONFIDENCE($\varphi, \mathcal{S}$)

Input: A dependency φ over a dataspace $\mathcal{S}$
Output: An estimated confidence measure $\hat{c}$

```
1: $T := \emptyset$
2: $E := \{(t_i, t_j) \in \mathcal{S} \mid (t_i, t_j) \asymp \text{LHS}(\varphi), (t_i, t_j) \not\asymp \text{RHS}(\varphi)\}$
3: $U :=$ objects in $\mathcal{S}$
4: while $U \neq \emptyset$ do
5:     $t := \arg\min_{u \in U} |\{w \mid w \in U, (w, u) \in E\}|$
6:     $T := T \cup \{t\}$
7:     $U := U - \{t\} \cup \{w \mid w \in U, (w, t) \in E\}$
8: return $|T|/|\mathcal{S}|$
```

Proposition V.2. *The greedy Algorithm 2 outputs an estimate $\hat{c}$ with an approximation ratio $(\Delta+2)/3$, that is, $3c/(\Delta+2) \leq \hat{c} \leq c$. The complexity is $O(n^2)$.*

B. Randomized Approach

The reedy algorithm still has to consider all the objects in a dataspace. It is desirable to evaluate only a small subset of objects upon which the estimated measure is still guaranteed by certain approximation bound with a high probability. Random sampling has been studied for estimating the error measure of approximate functional dependencies [19] and the confidence of approximate conditional functional dependencies [24]. Similarly, motivated by the randomized algorithm for the minimum vertex cover problem [31], we can also draw a subset of objects from $\mathcal{S}$ to estimate the error and confidence measures.

Preliminary: Due to the hardness in approximating error and confidence measures, it is difficult to insist on the relative performance guarantee in the randomized computation. Instead, we have an additive error $\epsilon, 0 \leq \epsilon \leq 1$, allowed upon the approximation ratio. That is, the approximation ratio with an additive error ϵ is guaranteed with a high probability at least δ. In order to locally compute measures with respect to a subset of objects in dataspaces, again, the number of violations of each object should be bounded, i.e., Δ.

Estimated Error: Suppose that V' is a (approximate) minimum violation set of objects in $\mathcal{S}$ with respect to φ, having $g' = |V'|/|\mathcal{S}|$. We uniformly and independently draw m objects from $\mathcal{S}$, where

$$m = \frac{2}{\epsilon^2} \log \frac{2}{1-\delta}$$

is determined by certain ϵ and δ as discussed below. Let X_i be a random variable with respect to object $t_i, 1 \leq i \leq m$, such that $X_i = 1$ if t_i belongs to the minimum violation set V'; otherwise, $X_i = 0$. We approximately estimate the error measure by

$$\hat{g} = \frac{1}{m} \sum_{1 \leq i \leq m} X_i + \frac{\epsilon}{2}. \tag{1}$$

Lemma V.1. *Let $m = \frac{2}{\epsilon^2} \log \frac{2}{1-\delta}$, we have*

$$\Pr[g' \leq \hat{g} \leq g' + \epsilon] \geq \delta,$$

where ϵ is an additive error and δ is a probability guarantee.

Proof: Let $g' = E[X_i]$. According to Chernoff bound,

$$\Pr\left[\frac{1}{m}\sum X_i \geq g' + \alpha\right] \leq e^{-2\alpha^2 m},$$

$$\Pr\left[\frac{1}{m}\sum X_i \leq g' - \alpha\right] \leq e^{-2\alpha^2 m},$$

we have

$$\Pr\left[g' - \alpha \leq \frac{1}{m}\sum X_i \leq g' + \alpha\right] \geq 1 - 2e^{-2\alpha^2 m}.$$

Let $\alpha = \frac{\epsilon}{2}$.

$$\Pr\left[g' \leq \frac{1}{m}\sum X_i + \frac{\epsilon}{2} \leq g' + \epsilon\right] \geq 1 - 2e^{-\frac{\epsilon^2 m}{2}}.$$

Since $m = \frac{2}{\epsilon^2} \log \frac{2}{1-\delta}$ and $\hat{g} = \frac{1}{m} \sum X_i + \frac{\epsilon}{2}$, we have

$$\Pr\left[g' \leq \hat{g} \leq g' + \epsilon\right] \geq \delta,$$

which completes the proof. ■

Now, the problem is to determine whether or not t_i belongs to minimum violation set V', i.e., X_i equals to 1 or 0. To solve it efficiently, we consider distributed approaches that can locally decide X_i by using a subset of objects in a certain radius with respect to violation relationships.

Note that, in a graph with degree of each vertex bounded by Δ, an approximate minimum vertex cover with performance ratio $(2\log\Delta + 1)$ can be achieved in $\log\Delta$ rounds [31]. In other words, only those objects with violation path to t_i in radius $\log\Delta$ will be retrieved, and the output is guaranteed with $g \leq g' \leq (2\log\Delta + 1)g$. The pseudo-code with details for estimated error is given in Algorithm 3.

Algorithm 3 Randomized ERROR($\varphi, \mathcal{S}, \epsilon, \delta$)

```
Input: A dependency $\varphi$ over a dataspace $\mathcal{S}$, additive error $\epsilon$,
    and probability guarantee $\delta$
Output: An estimated error measure $\hat{g}$
1:  $X :=$ SAMPLE($\epsilon, \delta, \mathcal{S}$)
2:  $\mathcal{T}$ =comparable objects from $\mathcal{S}$ in radius $\log\Delta$ w.r.t. each
    $t_i \in X$
3:  $V := \emptyset$
4:  $E := \{(t_i, t_j) \in \mathcal{T} \mid (t_i, t_j) \asymp$ LHS$(\varphi), (t_i, t_j) \not\asymp$
    RHS$(\varphi)\}$
5:  $U :=$ objects in $\mathcal{T}$
6:  for $i = 1$ to $\log\Delta$ do
7:      for each $u \in U$ such that $|\{w \mid w \in U, (w, u) \in E\}| \geq$
        $\Delta/2^i$ do
8:          $V := V \cup \{u\}$
9:          $U := U - \{u\}$
10:         $E := E - \{(w, u) \in E \mid w \in U\}$
11: return $|V \cap X|/|X| + \epsilon/2$
```

The randomized Algorithm 3 returns an estimated error $\hat{g}$. Specifically, $\mathcal{T}$ denotes the objects from $\mathcal{S}$ that are connected to m samples in X by comparison functions within radius $\log\Delta$. Since the number of violations to an object is bounded by Δ, it requires $O(\Delta^{\log\Delta}m)$ operations to greedily remove violations. Line 6-10 computes an approximate minimum violation set V of objects in $\mathcal{T}$, in $\log\Delta$ rounds, with relative performance bounded by $(2\log\Delta+1)$ [31]. Together with the probability guarantee on sampling in Lemma V.1, Proposition V.3 is concluded.

Proposition V.3. *The randomized Algorithm 3 outputs an estimate $\hat{g}$ with the probability*

$$\Pr[g \leq \hat{g} \leq (2\log\Delta + 1)g + \epsilon] \geq \delta.$$

The complexity is $O(\Delta^{\log\Delta}m)$, where $m = \frac{2}{\epsilon^2}\log\frac{2}{1-\delta}$.

Estimated Confidence: Similarly, we can have a randomized version for estimating confidence. Let T' be a (approximate) maximum keeping set, having $c' = |T'|/|\mathcal{S}|$. Let Y_i be a random variable such that $Y_i = 1$ if object t_i belongs to the maximum keeping set T'; otherwise $Y_i = 0$. The confidence is estimated by

$$\hat{c} = \frac{1}{m}\sum_{1\leq i\leq m} Y_i - \frac{\epsilon}{2}. \tag{2}$$

Lemma V.2. *Let $m = \frac{2}{\epsilon^2}\log\frac{2}{1-\delta}$, we have*

$$\Pr[c' - \epsilon \leq \hat{c} \leq c'] \geq \delta,$$

where ϵ is an additive error and δ is a probability guarantee.

Proof: Let $Y_i = 1 - X_i$ and $c' = E[Y_i]$. According to Lemma V.1, we have

$$\Pr\left[1 - c' \leq 1 - \frac{1}{m}\sum Y_i + \frac{\epsilon}{2} \leq 1 - c' + \epsilon\right] \geq 1 - 2e^{-\frac{\epsilon^2 m}{2}}.$$

With the same $m = \frac{2}{\epsilon^2}\log\frac{2}{1-\delta}$ and $\hat{c} = \frac{1}{m}\sum Y_i - \frac{\epsilon}{2}$,

$$\Pr[c' - \epsilon \leq \hat{c} \leq c'] \geq \delta,$$

the conclusion is proved. ∎

Now, we locally compute Y_i with respect to T'. It is known that, in a Δ-degree-bounded graph, an approximate maximum independent set with performance ratio $(\Delta+2)/3$ can be found in $\min(\Delta^4\log n, \Delta!)$ rounds [30]. That is, we have to traverse objects with violation path to t_i in radius $\min(\Delta^4\log n, \Delta!)$ such that the output is guaranteed with $3c/(\Delta + 2) \leq c' \leq c$. The pseudo-code with details for estimated confidence is given in Algorithm 4.

Algorithm 4 Randomized CONFIDENCE($\varphi, \mathcal{S}, \epsilon, \delta$)

```
Input: A dependency $\varphi$ over a dataspace $\mathcal{S}$, additive error $\epsilon$,
    and probability guarantee $\delta$
Output: An estimated confidence measure $\hat{c}$
1:  $Y :=$ SAMPLE($\epsilon, \delta, \mathcal{S}$)
2:  $\mathcal{T}$ =comparable objects from $\mathcal{S}$ in radius
    $\min(\Delta^4\log n, \Delta!)$ w.r.t. each $t_i \in Y$
3:  $T := \emptyset$
4:  $E := \{(t_i, t_j) \in \mathcal{T} \mid (t_i, t_j) \asymp$ LHS$(\varphi), (t_i, t_j) \not\asymp$
    RHS$(\varphi)\}$
5:  $U :=$ objects in $\mathcal{T}$
6:  while $U \neq \emptyset$ do
7:      for each $u \in U$ such that $d(u) \leq \frac{\sum_{(w,u)\in E} d(w)}{d(u)}$ do
8:          $T := T \cup \{u\}$
9:          $U := U - \{u\} \cup \{w \mid w \in U, (w, u) \in E\}$
10:         $E := E - \{(w, u) \in E\}$
11: return $|T \cap Y|/|Y| - \epsilon/2$
```

The randomized Algorithm 4 returns an estimated confidence $\hat{c}$. Similarly, line 6-10 greedily computes an approximate maximum keeping set T of objects in $\mathcal{T}$ corresponding to m samples in Y. According to the greedy analysis in [30], it is sufficient to move an object u to T if $d(u) \leq \frac{\sum_{(w,u)\in E} d(w)}{d(u)}$. The computation can be done in $\min(\Delta^4\log n, \Delta!)$ rounds [30]. Thereby, $\mathcal{T}$ includes the objects from $\mathcal{S}$ within radius $\min(\Delta^4\log n, \Delta!)$ w.r.t. comparison functions to m samples in Y. The algorithm complexity is thus $O(\Delta^{\min(\Delta^4\log n, \Delta^\Delta)}m)$. Combining the result in Lemma V.2, we have

Proposition V.4. *The randomized Algorithm 4 outputs an estimate $\hat{c}$ with the probability*

$$\Pr[3c/(\Delta + 2) - \epsilon \leq \hat{c} \leq c] \geq \delta.$$

The complexity is $O(\Delta^{\min(\Delta^4 \log n, \Delta^\Delta)} m)$*, where* $m = \frac{2}{\epsilon^2} \log \frac{2}{1-\delta}$.

Randomized vs. Distributed: Recall that dataspaces may collect from various sources. The randomized algorithm can naturally be deployed in a distributed environment. We randomly sample objects from different sources, locally compute X_i or Y_i for each object t_i in its corresponding source, and finally aggregate them together as a global estimation according to formula (1) or (2).

VI. Pay-as-you-go Discovery

Once the validation of dependencies is carefully investigated, a natural extension is to find all such valid dependencies in dataspaces which meet desired error/confidence requirements. It is known as the *discovery* problem. Recall that, comparable correspondences are practically identified in a *pay-as-you-go* style [3]. Therefore, the discovery of dependencies should be conducted in an incremental way as well. Intuitively, given a set Σ of currently discovered dependencies and a newly identified comparison functions $\theta(A_i, A_j)$, we can generate new dependencies w.r.t. $\theta(A_i, A_j)$.

In the following, we first give an overview of pay-as-you-go discovery. The approximate implication of CDs is then introduced, in order to avoid redundancy during the discovery. Finally, we present the incremental discovery algorithm.

Overview: When a comparison function $\theta(A_i, A_j)$ is identified, the way of their values to be compared is recognized, e.g., based on equality [11] or metric distance less than certain threshold [7]. Mechanisms have been well studied in schema matching [32] and reference reconciliation [33] for identifying comparable attributes and values, respectively. Since it is not the focus of our study, we consider the comparison function $\theta(A_i, A_j)$ as the input of our discovery algorithm.

The traditional dependency discovery problem is to find a canonical cover of all dependencies that hold in a given relation instance. Several algorithms have been proposed for FDs, such as TANE [22], [23] and FastFDs [34]. However, it is known that an output canonical cover of all FDs may have exponential size with respect to the number of attributes, no matter what discovery algorithm is used [35], [36]. As mentioned in Section IV, FD is one of the special cases of our comparable dependencies. Such exponential complexity carries over to the dependencies in dataspaces. Due to the extremely high dimensionality (e.g., thousands of attributes and comparison functions), it is highly non-trivial (if not impossible) to discover a canonical cover of all dependencies in a given dataspace.

Motivated by the idea of mining k-length itemsets in association rules, we study the k-length dependencies, which contain k comparison functions. Instead of all dependencies with arbitrary length, we can practically discover a subset of dependencies, which contain k or less comparison functions and satisfy a requirement η of error/confidence measure. It is notable that the concept of confidence is different between dependencies and association rules. The confidence of association rule is defined with respect to the frequency of itemsets, while the confidence of dependency is with respect to non-violations of objects, analogous to transactions in association rule mining.

The incremental process of discovery is described as: given a set Σ of dependencies on the current function set Θ in $\mathcal{S}$ and a new function θ introduced to the dataspace, to discover those dependencies with respect to θ and add them to Σ. The update of k-length dependencies addresses two aspects: first, finding dependencies with the new function as RHS; second, augmenting dependencies with length less than k. The search space of dependencies with respect to the new function θ is the combination of $k-1$ functions from Θ, with complexity $O(|\Theta|^{k-1})$. A naive approach is to evaluate all these candidates in $\mathcal{S}$ and return those dependencies which can satisfy the measure requirement η. Intuitively, some of these dependencies may imply others as illustrated below, i.e., redundant dependencies may exist in the returned answers.

Approximate Implication: Approximate inference of functional dependencies has been considered with respect to error measure [19]. It turns out that the error of a dependency obtained by applying an inference rule should be no larger than the premise dependencies. In the following, we investigate that such bounds of error and confidence can also be obtained when inferring dependencies on comparison functions in dataspaces, e.g., by augmentation.

Proposition VI.1. *Consider a dependency* φ_1 *over* $\mathcal{S}$*. Let*

$$\varphi_2: \quad \text{LHS}(\varphi_1) \wedge \theta(A_i, A_j) \rightarrow \text{RHS}(\varphi_1).$$

We have $g_3(\varphi_1, \mathcal{S}) \geq g_3(\varphi_2, \mathcal{S})$ *and* $\mathsf{conf}(\varphi_1, \mathcal{S}) \leq \mathsf{conf}(\varphi_2, \mathcal{S})$.

Proof: Let T_1^* be a maximum keeping set with respect to φ_1 in $\mathcal{S}$, i.e., $\mathsf{conf}(\varphi_1, \mathcal{S}) = \frac{|T_1^*|}{|\mathcal{S}|}$. First, for any two objects $t_i, t_j \in T_1^*$, we have $(t_i, t_j) \vDash \varphi_1$ according to the definition of keeping set. It follows $(t_i, t_j) \vDash \varphi_2$ referring to the augmentation rule. Thereby, a keeping set T_2 could be as large as T_1^*, $T_2 = T_1^*$. On the other hand, for any object $t_i \in \mathcal{S} \setminus T_1^*$, there must exist an object $t_j \in \mathcal{S}$ such that $(t_i, t_j) \asymp \text{LHS}(\varphi_1)$ and $(t_i, t_j) \not\asymp \text{RHS}(\varphi_1)$. Suppose that we have $(t_i, t_j) \not\asymp \theta(A_i, A_j)$. Since (t_i, t_j) do not agree on the left-hand-side of φ_2, they do not violate φ_2 either. If t_i does not violate φ_2 with any other objects, it could appear in a keeping set T_2 but not in T_1^*, i.e., $T_2 \supset T_1^*$. To sum up, we have $T_2^* \supseteq T_1^*$ and the conclusion is proved according to error/confidence definition. ■

Redundancy: By applying the implication properties, we can infer dependencies from others. Therefore, there exist redundancies among all the k-length dependencies. During the discovery, we would like to eliminate these redundant dependencies in Σ.

Let φ_1 be a dependency in Σ, i.e., $g_3(\varphi_1, \mathcal{S}) \leq \eta$. Let φ_2 be an argumentation with respect to the new function $\theta(A_i, A_j)$,

$$\varphi_2: \quad \text{LHS}(\varphi_1) \wedge \theta(A_i, A_j) \rightarrow \text{RHS}(\varphi_1).$$

We always have $g_3(\varphi_2, \mathcal{S}) \leq \eta$ as well. This conclusion is natural according to Proposition VI.1 having $g_3(\varphi_2, \mathcal{S}) \leq g_3(\varphi_1, \mathcal{S}) \leq \eta$. In other words, those dependencies, which are already discovered satisfying the measure requirement, could be ignored in the augmentation.

Moreover, this conclusion also enables the level-wise search which is often used in data mining and discovering FDs [22], [23]. Specifically, each level l denotes all the candidates of function subsets from Θ with size l, having $l \leq k$. Once a candidate in current level l is valid and add in Σ, then all the candidates expending it in following levels $l+1, \ldots, k$ must also be valid, and can be ignored in Σ as redundancy.

Algorithm: The pseudo-code for incremental discovery is given in Algorithm 5, with time complexity $O(|\Theta|^{k-1}\mathcal{V})$ where $O(\mathcal{V})$ is the validation cost of each dependency. Specifically, the discovery algorithm returns a new set of dependencies by considering a new function θ, based on the previous set Σ of dependencies on functions Θ over a dataspace $\mathcal{S}$. By calling LEVEL-WISE$(\Theta, k-1)$, it returns a sequence of function subsets (with length $\leq k-1$) from Θ in a level-wise order, i.e., a set always comes before all of its supersets. Possible redundancies are pruned in line 5, 10, 13, according to the augmentation property in Proposition VI.1.

Algorithm 5 Incremental DISCOVERY(Σ, η, θ)

```
Input: A set $\Sigma$ of dependencies on functions $\Theta$ over dataspace $\mathcal{S}$, a measure requirement $\eta$, and a new function $\theta$
Output: A new set $\Sigma$ of $k$-length dependencies
1: $L :=$ LEVEL-WISE$(\Theta, k-1)$
2: for each function set LHS in $L$ do
3:     if $g_3(\text{LHS} \to \theta, \mathcal{S}) \leq \eta$ then
4:         $\Sigma := \Sigma \cup \{\text{LHS} \to \theta\}$
5:         remove all the function sets in $L$ that are superset of LHS
6: for each function RHS in $\Theta$ do
7:     $L :=$ LEVEL-WISE$(\Theta, k-2)$
8:     for each function set LHS in $L$ do
9:         if LHS $\to$ RHS $\in \Sigma$ then
10:            remove all the function sets in $L$ that are superset of LHS
11:        else if $g_3(\text{LHS} \wedge \theta \to \text{RHS}, \mathcal{S}) \leq \eta$ then
12:            $\Sigma := \Sigma \cup \{\text{LHS} \wedge \theta \to \text{RHS}\}$
13:            remove all the function sets in $L$ that are superset of LHS
14: return $\Sigma$
```

VII. EXPERIMENTS

In this section, we report an extensive experimental evaluation of proposed mechanisms on two real data sets, Base and Wiki. Google Base[3] is a *very large, self-describing, semi-structured, heterogeneous* data collection. Each entry consists of several attributes with corresponding values and can be regarded as an object in dataspaces. Due to the heterogeneity of data, which are contributed by users around the world, the data set is extremely sparse. According to our observation, there are total 129 attributes in 10,000 objects, while most of these objects only have less than 10 attributes individually.

[3]http://base.google.com/

TABLE III
EXAMPLE OF DEPENDENCIES

	Dependency	Conf c
φ_1 :	$\theta(\mathsf{mpn}, \mathsf{upc}) \to \theta(\mathsf{id}, \mathsf{id})$	0.99
φ_2 :	$\theta(\mathsf{mpn}, \mathsf{mpn}) \to \theta(\mathsf{id}, \mathsf{id})$	1.00
φ_3 :	$\theta(\mathsf{mpn}, \mathsf{mpn}) \to \theta(\mathsf{imagelink}, \mathsf{imagelink})$	0.96

Another real data set of dataspaces is from Wikipedia[4], where each article usually has an object with some attributes and values to describe the basic structured information of the entry. The attributes of objects in different entries are various (e.g., 251 attributes in 10k objects), while each object may only contain a limited number of attributes (less than 10). Again, all these objects from heterogeneous sources form a huge spare dataspace in Wikipedia.

Table III illustrates examples of comparable dependencies in the Base data set. The attributes mpn and upc denote a pair of comparable attributes in the heterogeneous data, i.e., the manufacture product number and unified product code, respectively. The dependency φ_1 in Table III states that if two products have comparable mpn or upc, then their ids should be comparable as well (e.g., denote the same product). Such dependencies are also the examples of results discovered by Algorithm 5.

Given certain dependencies[5], our experiments in Section VII-A evaluate both the effectiveness and efficiency of various computation algorithms for error and confidence measures. Moreover, in Section VII-B, we present the performance of pay-as-you-go discovery. We adopt the widely used cosine similarity with q-grams [6] as the comparison operator. Similar results in the secondary Wiki data set may be omitted due to the limitation of space. All algorithms are implemented by Java. Experiments run on a machine with Intel Core 2 CPU (2.13 GHz) and 2 GB of memory.

A. Validation Evaluation

To evaluate the approximation computation of error and confidence, we mainly observe the relative performance of exact/approximate measures and the corresponding computation time cost.

As illustrated in Figure 1 (a) and 3 (a), the relative performance of error $(\hat{g}/g)$ is not as stable as that of confidence $(c/\hat{c})$. The underlining reason is that we consider dependencies which almost hold, with low error/high confidence. For example, the confidence of φ_3 in Table III is about 0.96. A slight difference between c and $\hat{c}$ (e.g., 0.97 and 0.96) will not affect the relative performance $(c/\hat{c})$ largely, while such small absolute difference appears significant in the relative

[4]http://www.wikipedia.org/

[5]e.g., in Table III discovered by our Algorithm 5.

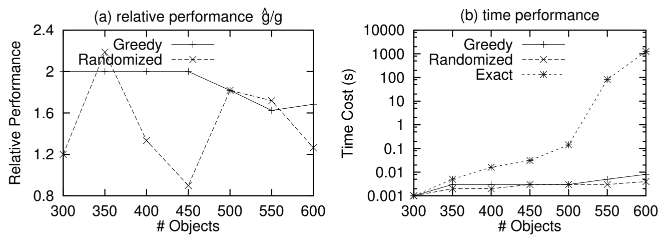

Fig. 1. Performance ($\hat{g}/g$, time) of error approximation

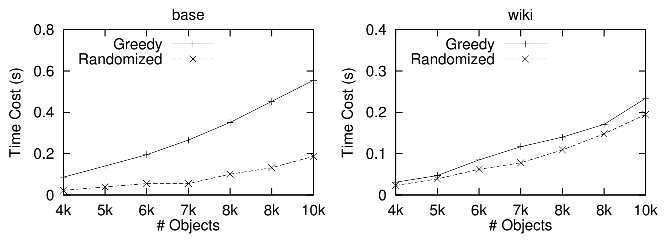

Fig. 2. Scalability of error approximation

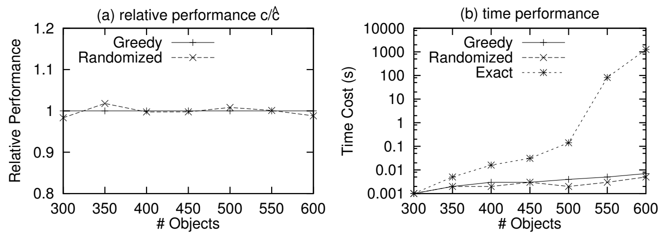

Fig. 3. Performance ($c/\hat{c}$, time) of confidence approximation

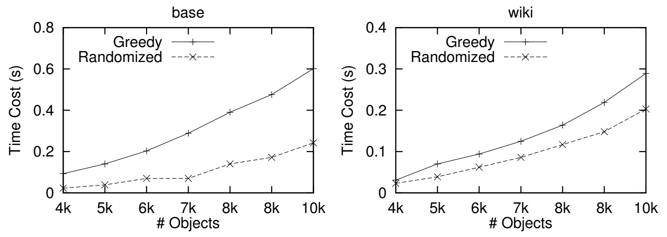

Fig. 4. Scalability of confidence approximation

performance of error measure, e.g., between 0.04 and 0.03 of $\hat{g}$ and g. For the randomized approach, we have requirements of additive $\epsilon = 0.2$ and probability $\delta = 0.8$. Although, the approximation performance might not be exactly bounded, e.g., under 350 objects in Figure 1 (a), it is bounded with high probability (guaranteed by $\delta = 0.8$).

Obviously, exact computation of validation does not scale well. As presented in Figure 1 (b) and 3 (b), the exact computation increases exponentially, which is not surprising according to our previous analysis of hardness in computing error and confidence. Meanwhile, the approximation computation keeps significantly lower time cost. To demonstrate the scalability of approximation computation, Figure 2 and 4 report the validation cost under various sizes of objects in dataspaces, in both Base and Wiki data sets.

Nevertheless, we also evaluate the randomized computation when different additive ϵ and probability δ are required. Figure 5 reports the sampling sizes under various requirements, and

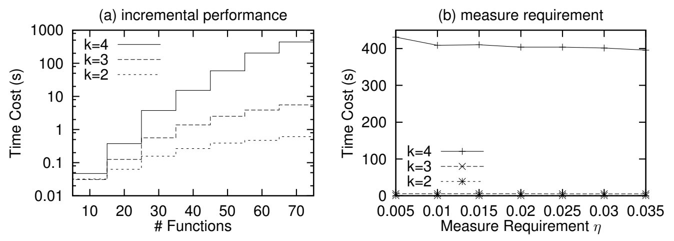

Fig. 6. Performance of pay-as-you-go discovery

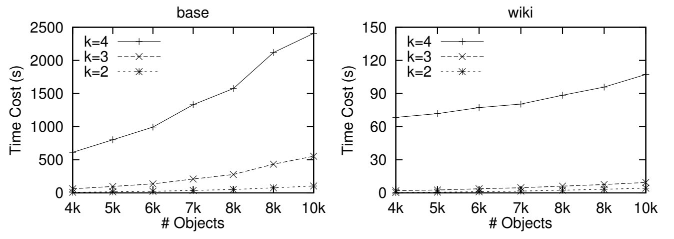

Fig. 7. Scalability of pay-as-you-go discovery

the corresponding effects on time costs. Generally, the larger the additive ϵ is allowed, the less the number of samples is required. On the other hand, to achieve higher probability δ guarantee, we need to draw more samples. Intuitively, using more samples will increase the computation costs, thereby the time costs of error and confidence in Figure 5 (b) and (c) are roughly affected by the size of samples.

B. Discovery Evaluation

In this experiment set, we evaluate the pay-as-you-go discovery of dependencies by calling Algorithm 5 incrementally, i.e., $\Sigma := \text{DISCOVERY}(\Sigma, \eta, \theta)$.

Figure 6 (a) illustrates the incremental discovery of dependencies with the increase of functions. It is notable that the y-axis is scaled logarithmically, that is, the time cost increases heavily with the number of functions. In fact, the size k of functions in a dependency also affects the discovery performance largely. It is not surprising due to the intrinsic hardness in discovering dependencies with respect to attributes (and the corresponding comparison functions). Although the measure requirement does not affect the performance significantly, a loose requirement like larger error requirement enables more pruning opportunities in Proposition VI.1. As presented in Figure 6 (b) with various error measure requirements η, a larger error requirement η needs less time cost.

As shown in Figure 7, the discovery algorithm scales well in large size of objects, since greedy approximation is adopted for validation. These results also verify the efficiency of approximation computation proposed in Section V for validating dependencies.

VIII. CONCLUSIONS

In this paper, we introduce data dependencies to dataspaces, namely *comparable dependencies* (CDs), which turn out practically useful in handling heterogeneous data. To our best knowledge, this is the first work to adapt dependencies to

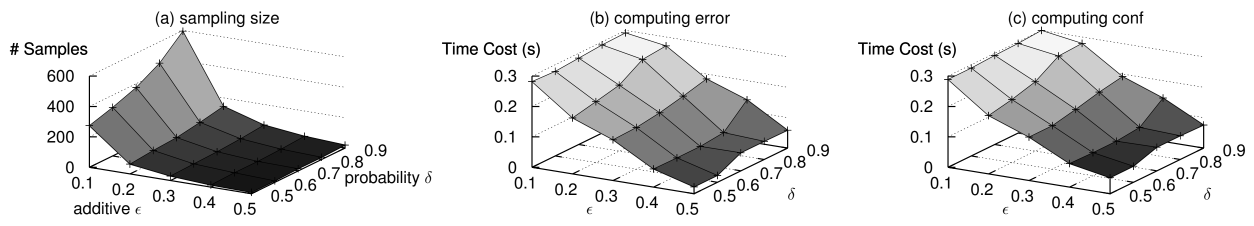

Fig. 5. Sampling size and time performance of randomized algorithm with various additive ϵ and probability δ

dataspaces with the consideration of comparable correspondences. Unfortunately, due to the heterogeneous data, as we proved, it is already hard to determine whether a dependency almost holds in the data. In fact, the confidence validation is also proved hard to approximate to within any constant factor. We propose several greedy and randomized approaches for approximately solving the validation problem. Our extensive experimental evaluation on real data sets demonstrates the performance of proposed greedy and randomized approaches. We believe that interesting studies can be raised on the proposed notations, some of which are still open. For example, a fundamental attempt would be a sound and complete set of inference rules for dependency implication in dataspaces, under certain premise of comparison functions.

Acknowledgments

This work is supported in part by Hong Kong RGC GRF 611608, US NSF through grants IIS-0905215, IIS-0914934, DBI-0960443, OISE-0968341 and OIA-0963278.

References

[1] A. Y. Halevy, M. J. Franklin, and D. Maier, "Principles of dataspace systems," in *PODS*, 2006, pp. 1–9.

[2] J. Madhavan, S. Cohen, X. L. Dong, A. Y. Halevy, S. R. Jeffery, D. Ko, and C. Yu, "Web-scale data integration: You can afford to pay as you go," in *CIDR*, 2007, pp. 342–350.

[3] S. R. Jeffery, M. J. Franklin, and A. Y. Halevy, "Pay-as-you-go user feedback for dataspace systems," in *SIGMOD Conference*, 2008, pp. 847–860.

[4] A. D. Sarma, X. Dong, and A. Y. Halevy, "Bootstrapping pay-as-you-go data integration systems," in *SIGMOD Conference*, 2008, pp. 861–874.

[5] M. A. V. Salles, J.-P. Dittrich, S. K. Karakashian, O. R. Girard, and L. Blunschi, "itrails: Pay-as-you-go information integration in dataspaces," in *VLDB*, 2007, pp. 663–674.

[6] G. Navarro, "A guided tour to approximate string matching," *ACM Comput. Surv.*, vol. 33, no. 1, pp. 31–88, 2001.

[7] W. Fan, J. Li, X. Jia, and S. Ma, "Reasoning about record matching rules," *PVLDB*, 2009.

[8] W. Fan, "Dependencies revisited for improving data quality," in *PODS*, 2008, pp. 159–170.

[9] A. Y. Levy and Y. Sagiv, "Semantic query optimization in datalog programs," in *PODS*, 1995, pp. 163–173.

[10] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis, "Conditional functional dependencies for data cleaning," in *ICDE*, 2007, pp. 746–755.

[11] D. Z. Wang, X. L. Dong, A. D. Sarma, M. J. Franklin, and A. Y. Halevy, "Functional dependency generation and applications in pay-as-you-go data integration systems," in *WebDB*, 2009.

[12] U. S. Chakravarthy, J. Grant, and J. Minker, "Logic-based approach to semantic query optimization," *ACM Trans. Database Syst.*, vol. 15, no. 2, pp. 162–207, 1990.

[13] J. Chomicki, "Semantic optimization techniques for preference queries," *Inf. Syst.*, vol. 32, no. 5, pp. 670–684, 2007.

[14] X. Dong and A. Y. Halevy, "Indexing dataspaces," in *SIGMOD Conference*, 2007, pp. 43–54.

[15] M. Arenas, L. E. Bertossi, and J. Chomicki, "Consistent query answers in inconsistent databases," in *PODS*, 1999, pp. 68–79.

[16] N. Koudas, A. Saha, D. Srivastava, and S. Venkatasubramanian, "Metric functional dependencies," in *ICDE*, 2009, pp. 1275–1278.

[17] L. Golab, H. J. Karloff, F. Korn, D. Srivastava, and B. Yu, "On generating near-optimal tableaux for conditional functional dependencies," *PVLDB*, vol. 1, no. 1, pp. 376–390, 2008.

[18] W. Fan, F. Geerts, L. V. S. Lakshmanan, and M. Xiong, "Discovering conditional functional dependencies," in *ICDE*, 2009, pp. 1231–1234.

[19] J. Kivinen and H. Mannila, "Approximate inference of functional dependencies from relations," *Theor. Comput. Sci.*, vol. 149, no. 1, pp. 129–149, 1995.

[20] I. F. Ilyas, V. Markl, P. J. Haas, P. Brown, and A. Aboulnaga, "Cords: Automatic discovery of correlations and soft functional dependencies," in *SIGMOD Conference*, 2004, pp. 647–658.

[21] M. A. V. Salles, J. Dittrich, and L. Blunschi, "Intensional associations in dataspaces," in *ICDE*, 2010.

[22] Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen, "Efficient discovery of functional and approximate dependencies using partitions," in *ICDE*, 1998, pp. 392–401.

[23] ——, "Tane: An efficient algorithm for discovering functional and approximate dependencies," *Comput. J.*, vol. 42, no. 2, pp. 100–111, 1999.

[24] G. Cormode, L. Golab, F. Korn, A. McGregor, D. Srivastava, and X. Zhang, "Estimating the confidence of conditional functional dependencies," in *SIGMOD Conference*, 2009, pp. 469–482.

[25] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher, Eds. Plenum Press, 1972, pp. 85–103.

[26] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[27] G. Karakostas, "A better approximation ratio for the vertex cover problem," *ACM Trans. Algorithms*, vol. 5, no. 4, pp. 1–8, 2009.

[28] I. Dinur and S. Safra, "The importance of being biased," in *STOC*, 2002, pp. 33–42.

[29] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy, "Approximating clique is almost np-complete (preliminary version)," in *FOCS*, 1991, pp. 2–12.

[30] M. M. Halldórsson and J. Radhakrishnan, "Greed is good: approximating independent sets in sparse and bounded-degree graphs," in *STOC*, 1994, pp. 439–448.

[31] M. Parnas and D. Ron, "Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms," *Theor. Comput. Sci.*, vol. 381, no. 1-3, pp. 183–196, 2007.

[32] E. Rahm and P. A. Bernstein, "A survey of approaches to automatic schema matching." *VLDB J.*, vol. 10, no. 4, pp. 334–350, 2001.

[33] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, "Duplicate record detection: A survey," *IEEE Trans. Knowl. Data Eng.*, vol. 19, no. 1, pp. 1–16, 2007.

[34] C. M. Wyss, C. Giannella, and E. L. Robertson, "Fastfds: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances - extended abstract," in *DaWaK*, 2001, pp. 101–110.

[35] H. Mannila and K.-J. Räihä, "Dependency inference," in *VLDB*, 1987, pp. 155–158.

[36] ——, "Algorithms for inferring functional dependencies from relations," *Data Knowl. Eng.*, vol. 12, no. 1, pp. 83–99, 1994.