The VLDB Journal (2013) 22:253–274
DOI 10.1007/s00778-012-0285-7

REGULAR PAPER

Comparable dependencies over heterogeneous data

Shaoxu Song · Lei Chen · Philip S. Yu

Received: 6 November 2011 / Revised: 23 April 2012 / Accepted: 27 June 2012 / Published online: 31 July 2012

Abstract To study the 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, such as semantic query optimization. Due to heterogeneous data in dataspaces, the first question, known as the validation problem, is to tell 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, such as 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.

Keywords Comparable dependencies · Dataspaces

S. Song
Key Laboratory for Information System Security,
Ministry of Education; TNList; School of Software,
Tsinghua University, Beijing, China
e-mail: sxsong@tsinghua.edu.cn

L. Chen (✉)
Department of Computer Science and Engineering,
The Hong Kong University of Science and Technology,
Clear Water Bay, Hong Kong
e-mail: leichen@cse.ust.hk

P. S. Yu
Department of Computer Science, University of Illinois at Chicago,
Chicago, IL, USA
e-mail: psyu@cs.uic.edu

P. S. Yu
Computer Science Department, King Abdulaziz University,
Jeddah, Saudi Arabia

1 Introduction

The importance of dataspace systems has already been recognized and emphasized in handling heterogeneous data [23,29,37,47,46]. 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 1.

Example 1 We consider a dataspace with 3 objects,

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

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

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

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 [29], for example, 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 to be comparable, for example, 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* [41]. Moreover, the matching correspondence [29] can also be identified between two elements that denote the same entity in real world, for example, red and cardinal are said to be *matched* as comparable color. It is usually confirmed by a *matching operator*, for example, via update with dynamic semantics [17] or users' feedback [29].

Data dependencies have already shown to be important in various data-oriented practice [15], such as optimizing query evaluation [36], capturing data inconsistency [5], removing data duplicates [17], etc. It is promising to study data dependencies for the heterogeneous data in dataspaces as well. Unfortunately, little work has been drawn to address such data dependencies. Although Wang et al. [53] 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, 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.

1.1 Motivation example

Intuitively, the comparable correspondence may occur either in 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 specified on two attributes (manu, prod) w.r.t. metric operators in Example 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 to be comparable on (manu, prod) if at least one of these three comparison operators in $\theta(\mathsf{manu}, \mathsf{prod})$ evaluates to true. For example, (t_1, t_2) are comparable on (manu, prod), since the edit distance of $(t_1[\mathsf{manu}], t_2[\mathsf{prod}])$ is ≤ 5. Similarly, (t_1, t_3) are also comparable on (manu, prod), where $(t_1[\mathsf{manu}], t_3[\mathsf{manu}])$ satisfy '$\mathsf{manu} \approx_{\leq 5} \mathsf{manu}$'. Note that the edit distance thresholds in different attribute pairs in a comparable function are not necessary to be the same, and the identification of such comparable functions is not the focus of this study.

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 [15], comparable dependencies are also promising for dataspace applications.

Indeed, the most important motivation of this study is to improve the dataspace query efficiency [51], in particular with the proposed comparable dependencies. Recall that, in semantic query optimization [36], a conjunctive query can be rewritten by using part of predicates according to the data dependencies. Such optimization can also be introduced to queries in dataspaces on comparable attributes. For example, we consider a query object with (post : InfiniteLoop, CA) and (manu : Apple). The query evaluation [13] 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 (see details in Sect. 7).

1.2 Applications

Besides optimizing dataspace queries, we believe that the proposed comparable dependencies can meet a wide range of applications as the previous integrity constraints. As it is not the major focus of this study, we briefly describe several applications below and leave more details as future work.

Violation Detection Due to heterogeneous data, detecting violations is urgent to enhance the utility of dataspaces. Previously, data dependencies were often employed to handle violations in data [5,34], that is, those objects not satisfying data dependencies. Given a set Σ of dependencies, the violation detection problem is to find a minimum set of objects from the data (which do not satisfy the given dependencies) such that the dependencies in Σ hold in the remaining data. Unfortunately, as we proved in Sect. 4, it is already hard to find a

minimum violation set with respect to a single comparable dependency. The violation detection over a set Σ of CDs is thus highly non-trivial.

Consistent Query Answering Methods are proposed to remove the violations by repairing [15]. However, dataspaces often collect data from various sources without permission to write (repair). Thus, it is particularly interesting to study consistent query answering [1], which can return objects that have no violations without updating original data in dataspaces. The complexity of consistent query answering problem is determined by repair model, constraint language and query language. In dataspaces scenario, given dependencies with general comparison functions, the X-repair [10] by removing a minimum set of violation objects is already hard. Moreover, as stated in the aforesaid motivation example, a query over dataspaces not only considers the attributes specified in the query but also expands to other attributes according to comparison functions, which makes the consistent query answering more challenging.

Object Identification Since data are collected from various sources, it is not surprising that there are many duplicates in dataspaces. When matching operator is adopted in the right-hand-side, a matching dependency is considered. Matching dependencies (MDs) [15] and the corresponding deduction techniques [17] are proved useful in identifying duplicate objects (see [14] for a survey) in traditional databases. As one type of comparable dependencies we considered in dataspaces, MDs are naturally applicable to identify duplicate objects in dataspaces as well.

1.3 Challenges and contributions

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 tell whether a dependency approximately hold in a dataspace, that is, 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 with respect to the newly identified comparison function is urgent and not considered in previous dependency discovery.

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 dependencies well known in databases, such as functional dependencies, metric functional dependencies [34] and matching dependencies [15]. The widely used error and confidence measures are introduced to evaluate how a dependency approximately holds in a dataspace.

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

(iii) We develop approximation computation of 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 unless P = NP. We develop greedy approaches with approximation ratio on the bound of violations that an object may introduce. Moreover, randomized approaches are further developed to improve the efficiency, where approximation bound with an additive error is obtained with 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 that incrementally discovers data dependencies with respect to the newly identified comparison function.

(v) We study the semantic query optimization in dataspaces with comparable dependencies. As mentioned, according to the semantics of data dependencies, we can safely eliminate query predicates. Two types of predicate elimination techniques are presented. It is worth noting that applying approximate dependencies in the query rewriting will never lose query answers.

(vi) Finally, we report an extensive experimental evaluation of proposed approaches on real data sets. Both the effectiveness and efficiency of proposed approximation computation techniques are illustrated. The discovery performance is also evaluated on real data sets. Moreover, we demonstrate the accuracy and time performance of query optimization by using the returned comparable dependencies.

The remainder of this paper is organized as follows. First, we discuss the related work in Sect. 2. Then, in Sect. 3,

we study the foundations of comparable dependencies in dataspaces. Section 4 introduces the validation problem of approximate dependencies. In Sect. 5, we develop efficient approaches for computing error and confidence measures. Section 6 studies the pay-as-you-go discovery of comparable dependencies in dataspaces. Section 7 presents the techniques for optimizing dataspace queries. In Sect. 8, we report our extensive experimental results. Finally, Sect. 9 concludes this paper. An early version of this work appears in [50].

2 Related work

Although data dependencies have been well studied in databases, little work was drawn over heterogeneous data, especially in dataspaces with comparable correspondences. Table 1 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 [5] 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 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 [16,22]. 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. [53] extends functional dependencies with probability for data integration systems, namely *probabilistic functional dependencies* (pFDs), which is similar to the concepts of approximate FDs [33] and soft FDs [28]. 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. [34] 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 $\mathsf{name} \rightarrow^{9} \mathsf{addr}$. In our work, we adapt such dependencies with metrics to dataspaces by introducing comparison functions. Note that techniques in [34] only verify whether or not an MFD exactly holds, that is, exact MFDs, while the validation problem is not studied to tell whether a dependency almost/approximately holds with certain error/confidence guarantee, that is, approximate MFDs. As we investigated in special cases (Sect. 4), the validation problem of approximate MFDs in databases is NP-complete, which is not addressed in previous works.

Song and Chen [48] introduce *differential dependencies* (DDs), which employ metric operators on both sides of attributes in the dependency. Thereby, DDs can also be interpreted as a generalization of MFDs. The basic idea of DDs is: based on the metric distance constraints on certain attributes X, we can imply the corresponding metric distance constraints on the attributes Y. The determination of distance threshold for metric operators is also studied [49]. Again, since DDs consider the metric operator in an attribute in databases, the comparable correspondence among heterogeneous attributes in dataspaces are not addressed.

Fan [15] 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 [17].

Table 1 Related work

Dependencies	Operators	Comparable correspondences	Measures	Validation
pFDs [53]	Equality operator	Cannot address	Probability	PTIME
CFDs [5]	Equality operator	Cannot address	Confidence and condition support	PTIME
MFDs [34]	Equality and metric operators	Cannot address	Not studied	Not studied
MDs [15]	Metric and matching operators	Cannot address	Not studied	Not studied
Our CDs	All the above operators	Address	Error or confidence	NP-complete

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.

Measures are necessary during the discovery to evaluate how a dependency holds in a relation instance [22,26,27]. For evaluating FDs, the g_3 measure [33] is widely used, that is, the minimum number of tuples that have to be removed from the relation instance for the FD to hold. It can be efficiently computed by grouping tuples according to equal values. This error measure and its variations are widely used in discovering approximate functional dependencies [27,32] and evaluating CFDs [16,22]. In this study, we also employ the error measure to validate comparable dependencies in dataspaces which is, however, not easy to compute. Besides the error measure, Pfahringer and Kramer [35,43] propose a measure based on the minimum description length principle. An encoding length of a table T is defined based on $X \rightarrow Y$ to compress T. Giannella and Robertson [21] develop an approximation measure of FDs based on the intuition: the degree to which $X \rightarrow Y$ is approximate in a table T is the degree to which T determines a function from $\Pi_X(T)$ to $\Pi_Y(T)$. In addition, Chiang and Miller [8] also study some other measures such as conviction and χ^2-test for evaluating CFDs. In our study, since we consider comparable correspondence in tuple pairs instead of groups of tuples on equality function, the above measures defined on equal values are not appropriate for evaluating CDs.

The discovery of dependencies from a given relation instance is widely studied [4,35,39,40]. In discovering FDs, previous work targeted on generating a canonical cover of all FDs. Huhtala et al. [26,27] propose a level-wise algorithm, namely TANE, together with efficient pruning when searching in the lattice of attributes. Remarkably, the TANE algorithm also supports the discovery of approximate FDs. Wyss et al. [54] study a depth-first, heuristic-driven algorithm, namely FastFDs, which is (almost) linear to the size of FDs cover. Flach and Savnik [19] discover FDs in a bottom-up style, which considers the maximal invalid dependencies first. When searching in a hypotheses space, the maximum invalid dependencies are used for pruning the search space. In discovering CFDs, Chiang and Miller [8] explore CFDs by considering all the possible dependency rules when $X \rightarrow Y$ is not specified. In [16], Fan et al. also study the case when the embedded FDs are not given and propose algorithms for different scenarios. When a rule $X \rightarrow Y$ is suggested, Golab et al. [22] study the discovery of optimal CFDs with the minimum pattern tableau size. A concise set of patterns are naturally desirable that may have lower cost in applications such as violation detection by CFDs. Unfortunately, all the above previous works on dependency discovery consider equality function without considering the comparable correspondence. Different from FDs and CFDs, there are various comparison functions that can be specified on the comparable attributes in CDs. Recognizing the hardness of discovering a canonical cover of all dependencies in a given dataspace, we focus on practically returning a subset of dependencies within a certain length.

The major difference of querying in dataspaces originates from the heterogeneous attributes with comparable correspondence, which prevents directly applying the previous database tools to optimize the dataspace queries [51]. Typical semantic query optimization techniques include predicate elimination, predicate introduction, join elimination and so on [7]. Based on the features of dataspaces queries, in this work, we mainly study the predicate elimination with comparable dependencies. Most previous works of semantic query optimization focus the theoretical aspects [6,9,36]. In order to make semantic query optimization useful in practice, many integrity constraints should be defined for a given database. Otherwise, only few queries could be optimized semantically. Consequently, in the practice of experimental evaluation, e.g., in [7,25,52], representative sample queries are usually considered, where the given integrity constraints can be applied. The experiments compare the query performance with and without semantic query optimization by such integrity constraints. Following this convention of representative sample queries, our experimental study evaluates the trade-off between the query time performance and the query accuracy performance, by applying different approximate dependencies.

The application of detecting violations is also known as a type of data repairing, that is, X-repair via tuple deletion [10]. When given a single FD or CFD, it is easy to find a minimum set of tuples to delete in order to ensure the remaining data satisfying the constraint. This minimum violation set computation is also known as the validation evaluation of a dependency over a dataset, as studied in Table 1. Unfortunately, for the CDs studied in our paper, we have shown that computing the minimum violation set for a single CD is already hard. When given a set of data dependencies instead of a single one, repairing w.r.t. a set of CFDs is NP-complete [5], while repairing w.r.t. denial constraints on numerical attributes is MAXSNP-hard [3]. As CDs appear harder than CFDs in the single dependency case, it is not surprising that repairing w.r.t. a set of CDs could be no easier than that of CFDs. In our theoretical analysis results, we have constructed the connections between the CDs violations and the general notation of graph. As future work, by leveraging the relationships between CDs on heterogeneous attributes and denial constraints on numerical attributes, it is interesting to study the potential of applying the proposed approximation and randomization techniques to repairing on numerical values.

Table 2 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

3 Foundations

In this section, we address the fundamental issues of adapting data dependencies to the heterogeneous data in dataspaces. Table 2 lists the frequently used notations.

3.1 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 true if $a_i = a_j$, that is, identical.
- $\leftrightarrow_{ij}$ can also be a *metric operator* $A_i \approx_\lambda A_j$, which is raised in metric functional dependencies [34]. Let d_{ij} be a distance metric[1] defined on the domains of two comparable attributes A_i, A_j. The metric operator $\approx_\lambda$ evaluates to true if $d_{ij}(a_i, a_j) \leq \lambda$, that is, 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 [15]. The matching operator $\rightleftharpoons$ evaluates to true if a_i and a_j are identified as matched, that is, make them identical via dynamic semantics [15] or by users' feedback [29].

The comparison operator $\leftrightarrow_{ij}$ can have but is not limited to the semantics mentioned above. In general, we assume that the comparison operator is commutative, that is, $A_i \leftrightarrow_{ij} A_j$ and $A_j \leftrightarrow_{ij} A_i$ are equivalent. However, it is not necessary to be idempotent, as two values from the same attribute A_i might not always evaluate to true. For each pair of attributes A_i, A_j in $\mathcal{S}$, we consider **one** comparison operator on them.

3.1.1 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 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 $\theta(A_i, A_j)$, that is,

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

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

For instance, we have $(t_1, t_2) \asymp \theta(\text{manu}, \text{prod})$ in Example 1, since the edit distance between $t_1[\text{manu}]$ and $t_2[\text{prod}]$ 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, for example, among (addr, street, city). Since the comparison operator is not the focus of this study, we will leave such comparison operators as our future work.

3.1.2 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 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 $(t_1, t_2) \asymp \theta(B_1, B_2)$.

[1] For example, the absolute value of difference on numerical values, or edit distance on string values (see [41] for a survey).

[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 presented below.

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 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 $(t_1, t_2) \vDash \varphi$, that is, any two objects in $\mathcal{S}$ satisfy φ.

Example 2 Consider the dataspace $\mathcal{S}$ as illustrated in Example 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 any two product objects have comparable (manu, prod) and comparable (addr, post), then their website should be matched, for example, itunes.com and apple.com are matched URLs of the same web site.

It is notable that comparable dependencies are different from the concept of tail in dataspaces [45,46]. 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.

3.2 Approximate dependencies

Due to the extremely high heterogeneity, data dependencies might not exactly hold in a given dataspace, that is, 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 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 [26,33].

3.2.1 Error measure

To evaluate how a dependency almost/approximately holds in a data instance, g_3 error measure [33] is widely used [26,27]. We can also adopt this measure to evaluate a dependency φ over a dataspace $\mathcal{S}$.

Definition 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, that is,

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

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 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 $g_3 = 1/3$.

3.2.2 Confidence measure

Instead of measuring violations, a confidence [11,22] defines the proportion of tuples after removing minimum tuples (objects in dataspace) of violations w.r.t. φ.

Definition 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 [53], this confidence measure is close to the concept of probability of FDs, which also consider 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, that is, non-violation.

For instance, $\{t_1, t_3\}$ in Example 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.

4 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 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. More precisely, 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 $\mathsf{conf}(\varphi, \mathcal{S}) \geq \eta_c$.

4.1 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 [11,27]. 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, that is, 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 3 for instance. The efficient validation computation based on disjoint grouping [27] cannot be applied to this case with general functions. In fact, we can prove the hardness of validation problem.

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

Proof The error/confidence validation problem is clearly in NP: given a set W of objects in $\mathcal{S}$, it is easy to check whether W is a violation set/keeping set with respect to a dependency such that $\frac{|W|}{|\mathcal{S}|}$ satisfies the measure requirement η.

To show that the error validation problem is NP-hard, we exhibit a polynomial-time reduction from the VERTEX COVER problem, which is known as one of Karp's 21 NP-complete problems [31]. Given a graph $\mathcal{G}(U, E)$ with $n = |U|$ vertices and $m = |E|$ edges, a vertex cover is a subset $C \subseteq U$ of vertices such that for each edge $(u_i, u_j) \in E$, C contains at least one of u_i or u_j. Let $\varphi : \theta(A_1, A_2) \rightarrow \theta(B_1, B_2)$ be a dependency where $\theta(A_1, A_2) : [A_1 \approx_{\leq m-1} A_1, A_1 \approx_{\leq m-1} A_2, A_2 \approx_{\leq m-1} A_2]$ specifies a constraint of the edit distance ($\leq m-1$) between the values of A_1 or A_2. We build a dataspace $\mathcal{S}$ where each object t_i corresponds to a vertex u_i in $\mathcal{G}$. For each object t_i, assign $t_i[B_1] = t_i[B_2] = b_i$ such that $(b_i, b_j) \not\asymp \theta(B_1, B_2), i \neq j, \forall i, j \in [1, n]$. Moreover, $t_i[A_1]$ and $t_i[A_2]$ values populate a string in length m, where all the string elements are a_i by default, that is, for all kth element $t_i[A_1][k] = t_i[A_2][k] = a_i, k \in [1, m]$. Similarly, we have $a_i \neq a_j, i \neq j, \forall i, j \in [1, n]$. Now, for each edge $e_k = (u_i, u_j) \in E, k \in [1, m]$, replace the kth element by c_k in the strings of attribute A_1, A_2 corresponding to u_i, u_j, respectively. That is, assign $t_i[A_1][k] = t_i[A_2][k] = t_j[A_1][k] = t_j[A_2][k] = c_k$ where c_k are different elements for different $k \in [1, m]$. The transformation completes and can be done in polynomial time.

We can see that the graph has a vertex cover C of size $|C| \leq \eta|U|$ if and only if there is a violation set V having $|V| \leq \eta|\mathcal{S}|$ objects with respect to the dependency $\varphi : \theta(A_1, A_2) \rightarrow \theta(B_1, B_2)$. Recall that a violation set V ensures that all the remaining objects satisfy the dependency, that is, $\mathcal{S} \setminus V \vDash \varphi$.

First, let C be a vertex cover of size $\eta|U|$. For each edge $e_k = (u_i, u_j) \in E$ with $u_i \in C$, we have $t_i[A_1][k] = t_i[A_2][k] = t_j[A_1][k] = t_j[A_2][k] = c_k$. In other words, edit distances of $(t_i[A_1], t_j[A_2])$ and $(t_i[A_2], t_j[A_1])$ agree on $\leq m-1$, that is, $(t_i, t_j) \asymp \theta(A_1, A_2)$. Recall that we have $(t_i, t_j) \not\asymp \theta(B_1, B_2)$ according to the assignment of values on B_1, B_2. Thereby, we can add t_i into the violation set V in order to eliminate $(t_i, t_j) \nvDash \varphi$. On the other hand, for any $u_i, u_j \notin C$, there should be no edge connecting them. Considering the l-th elements in corresponding objects $t_i, t_j, l \in [1, m]$, $t_i[A_1][l]$ and $t_i[A_2][l]$ could be either a_i or c_l, while $t_j[A_1][l]$ and $t_j[A_2][l]$ could be either a_j or c_l. However, t_i and t_j cannot have c_l at the same time, since e_l denotes an edge, which would not connect u_i and u_j. In other words, the edit distances of $(t_i[A_1], t_j[A_2])$ and $(t_i[A_2], t_j[A_1])$ equal to m which does not agree on $\leq m-1$, that is, $(t_i, t_j) \not\asymp \theta(A_1, A_2)$. Consequently, we have $(t_i, t_j) \vDash \varphi$ which do not belong to the violation set V. Clearly, we have $|V| = \eta|\mathcal{S}| = \eta|U|$.

Conversely, we can show that if there is a violation set V of size $|V| \leq \eta|\mathcal{S}|$, then we must have a vertex cover C of size $|C| \leq \eta|U|$. Suppose that the graph has is a minimum vertex cover C^* of size $|C^*| > \eta|U|$. Let C be the set of vertices corresponding to the objects in V. Obviously, C is not a vertex cover since $|C| = |V| < |C^*|$, and there should be at least $|C^*| - |C|$ edges connecting between different vertices in $C^* \setminus C$ and $U \setminus C^*$. For each edge $e_k = (u_i, u_j) \in E, u_i \in C^* \setminus C, u_j \in U \setminus C^*$, their corresponding (t_i, t_j) will introducing a violation with respect to φ. Thereby, we have $|V| \geq |C| + (|C^*| - |C|) = |C^*| > \eta|U| = \eta|\mathcal{S}|$, which is a contradiction.

To sum up, the error validation problem is proved to be NP-complete. Note that a violation set V for error measure is the complement of the keeping set T for confidence measure. Thereby, to prove the hardness of confidence validation

problem, we can build a reduction from the CLIQUE problem, which follows similar steps as shown in the proof of Theorem 2.

Although the proof utilizes the original (general) notation of comparison function $\theta(A_1, A_2)$, without loss of generality, it is sufficient to complete the proof by the simple $\theta(A_1, A_1)$ on the same attribute. □

It is worth noting that the above conclusion is not only valid for CDs in dataspaces but also valid for MDs in databases, where distance metric and matching identification are applicable. Intuitively, the transitivity is not valid either on the metric operators used in MDs. Indeed, our proof of Theorem 1 is sufficient to show that the validation of MDs is also NP-complete.

4.2 Special case of aligned attributes

An interesting special case is to consider comparison functions on aligned attributes, that is, $\theta(A_i, A_i)$. 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 3 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

$$t_1 : \{(A_1 : abc), \ldots\};$$
$$t_2 : \{(A_1 : abcd), \ldots\};$$
$$t_3 : \{(A_1 : abcde), \ldots\}.$$

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, $(t_1, t_2) \asymp \theta(A_1, A_1)$ and $(t_2, t_3) \asymp \theta(A_1, A_1)$ 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 2 *The error/confidence validation problem with aligned attributes in general comparison functions is still* NP*-complete.*

Proof To prove that the confidence validation problem with aligned attributes is NP-hard, we build a polynomial-time reduction from the CLIQUE problem. Given a graph $\mathcal{G}(U, E)$ with $n = |U|$ vertices and $m = |E|$ edges, a clique is a complete subgraph of $\mathcal{G}$, that is, a subset $Q \subseteq U$ of vertices such that $(u_i, u_j) \in E, \forall u_i, u_j \in Q$. Let $\varphi : \theta(A, A) \to \theta(B, B)$ be a dependency with comparison functions on aligned attributes where $\theta(B, B) : [B \approx_{\leq m-1} B]$ specifies a constraint of the edit distance ($\leq m-1$) between the values of B. We build a dataspace $\mathcal{S}$ where each object t_i corresponds to a vertex u_i in $\mathcal{G}$. For each object t_i, assign $t_i[A] = a$ such that $(t_i, t_j) \asymp \theta(A, A), i \neq j, \forall i, j \in [1, n]$. Moreover, $t_i[B]$ values populate a string in length m, where all the string elements are b_i by default, that is, for all kth element $t_i[B][k] = b_i, k \in [1, m]$. We have $b_i \neq b_j, i \neq j, \forall i, j \in [1, n]$. Now, for each edge $e_k = (u_i, u_j) \in E, k \in [1, m]$, replace the kth element by c_k in the strings of attribute B corresponding to u_i, u_j, respectively. That is, assign $t_i[B][k] = t_j[B][k] = c_k$ where c_k are different elements for different $k \in [1, m]$. The transformation completes and can be done in polynomial time.

Now, we must show that the graph has a clique Q of size $|Q| \geq \eta|U|$ if and only if there is a keeping set T having $|T| \geq \eta|\mathcal{S}|$ objects with respect to the dependency $\varphi : \theta(A, A) \to \theta(B, B)$. Recall that a keeping set T ensures that all the objects in T satisfy the dependency, that is, $T \vDash \varphi$.

Let Q be a clique of size $\eta|U|$. First, for any vertex $u_i \notin Q$, there must exist a vertex $u_j \in Q$ such that no edge connecting them, that is, $(u_i, u_j) \notin E$. Considering the l-th elements in corresponding objects $t_i, t_j, l \in [1, m]$, $t_i[B][l]$ could be either b_i or c_l, while $t_j[B][l]$ could be either b_j or c_l. However, t_i and t_j cannot have c_l at the same time, since e_l denotes an edge, which would not connect u_i and u_j. In other words, the edit distances of $(t_i[B], t_j[B])$ equals to m which does not agree on $\leq m-1$, that is, $(t_i, t_j) \not\asymp \theta(B, B)$. Recall that we have $(t_i, t_j) \asymp \theta(A, A), \forall t_i, t_j$. Consequently, $(t_i, t_j) \not\vDash \varphi$ implies that t_i does not belong to the keeping set T. On the other hand, for each vertex $u_i \in Q$, we have edges $e_k = (u_i, u_j) \in E, \forall u_j \in Q$. According to the above assignment of values on attribute B, we have $t_i[B][k] = t_j[B][k] = c_k$. In other words, we have edit distance of $(t_i[B], t_j[B])$ agreeing on $\leq m-1$, that is, $(t_i, t_j) \asymp \theta(B, B)$. It follows $(t_i, t_j) \vDash \varphi$ for all t_j corresponding to u_j in Q. Thereby, we can add t_i into the keeping set T as well. Consequently, we have $|T| = \eta|\mathcal{S}| = \eta|U| = |Q|$.

Next, we can show that if there is a keeping set T of size $|T| \geq \eta|\mathcal{S}|$, then we must have a clique Q of size $|Q| \geq \eta|U|$. Suppose that the graph has is a maximum clique Q^* of size $|Q^*| < \eta|U|$. Let Q be the set of vertices corresponding to the objects in T. Obviously, Q is not a clique since $|Q| = |T| > |Q^*|$, and there should be at least $|Q| - |Q^*|$ vertices in $Q \setminus Q^*$ which are not connected to a different vertex in Q^*, respectively. For each pair $(u_i, u_j) \notin E, u_i \in Q \setminus Q^*, u_j \in Q^*$, their corresponding (t_i, t_j) will introducing a violation with respect to φ. Thereby, we have to remove them from keeping set such that $|T| \leq |Q| - (|Q| - |Q^*|) = |Q^*| < \eta|U| = \eta|\mathcal{S}|$, which is a contradiction.

To sum up, the confidence validation problem with aligned attributes is NP-complete. The proof of error validation

problem follows a similar reduction from the VERTEX COVER problem as shown in the proof of Theorem 1. □

Moreover, our proof of Theorem 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.

4.3 Tractable special case

Finally, another special case with both aligned attributes and equality function, that is, $[A_i = A_i]$, is exactly the case of FDs with equality in traditional databases. As mentioned, such a special case has been well investigated with efficient algorithms [27].

5 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 2 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, for example, $\hat{g}/g \leq \rho$ 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, for example, $\Pr[\hat{g} \leq \rho g + \epsilon] \geq \delta$, where ϵ is an additive error and δ is a probability guarantee.

In Sect. 5.1, 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 Sect. 5.2, 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.

5.1 Greedy approach

Preliminary Given a comparison function $\theta(A_i, A_j)$, we can obtain a set of object pairs that agree on the function, that is, $\{(t_i, t_j) \in \mathcal{S} \mid (t_i, t_j) \asymp \theta(A_i, A_j)\}$. 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, that is, $L \setminus H$. Based on these object pairs of violations, the error and confidence measures can be computed.

5.1.1 Approximate error

We first present the approximation computation of g_3 error. According to the proof of Theorem 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 [20], by finding a maximal matching in the graph.

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 [20]. 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(φ, $\mathcal{S}$)

Require: A dependency φ over a dataspace $\mathcal{S}$
Ensure: 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 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, for example, with an approximation factor of $2 - \Theta(\frac{1}{\sqrt{\log n}})$ in [30], we do not consider them here. In fact, due to the hardness of approximating minimum vertex cover problem [12], it is not surprising to have:

Theorem 3 *The g_3 error is* NP*-hard to approximate to within any factor* $10\sqrt{5} - 21 \approx 1.36067$.

Proof We show that the reduction from the VERTEX COVER problem in the proof of Theorem 1 is a *gap-preserving* reduc-

tion. As shown in the proof of Theorem 1, a graph $\mathcal{G}(U, E)$ has a vertex cover C of size $\eta|U|$ if and only if the dataspace $\mathcal{S}$ has a violation set V of size $\eta|\mathcal{S}|$ for the error measure with respect to dependency φ. Let $C^*(\mathcal{G})$ and $V^*(\varphi, \mathcal{S})$ denote a minimum vertex cover for $\mathcal{G}$ and a minimum violation set for dataspace $\mathcal{S}$ w.r.t. φ, respectively. Recall that the exact error measure is $g_3(\varphi, \mathcal{S}) = |V^*(\varphi, \mathcal{S})|/|\mathcal{S}|$. If any approximation computes an approximate error measure $\hat{g} = |V|/|\mathcal{S}|$ with $\hat{g} \leq \alpha \cdot g_3(\varphi, \mathcal{S})$ for some constant $\alpha > 1$, it will produce a vertex cover C such that $|C| \leq \alpha \cdot |C^*(\mathcal{G})|$. More precisely, the reduction is gap-preserving:

- if $|C^*(\mathcal{G})| \leq \hat{g}|U|$, then $g_3(\varphi, \mathcal{S}) \leq \hat{g}$,
- if $|C^*(\mathcal{G})| > \alpha \cdot \hat{g}|U|$, then $|g_3(\varphi, \mathcal{S})| > \alpha \cdot \hat{g}$,

It is known that MINIMUM VERTEX COVER cannot be approximated within a factor of $\alpha = 10\sqrt{5} - 21 \approx 1.36067$ unless P = NP [12]. Thus, the error measure is NP-hard to approximate to within any constant factor less than $10\sqrt{5} - 21 \approx 1.36067$. □

5.1.2 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 4 *The confidence has no constant-factor approximation unless* P=NP.

Proof To show the hardness of confidence approximation, we give a gap-preserving reduction from the CLIQUE problem. Proofs of Theorems 1 and 2 show a reduction that a graph $\mathcal{G}(U, E)$ has a clique Q of size $\eta|U|$ iff there is a keeping set T of size $\eta|\mathcal{S}|$ with respect to dependency φ in dataspace $\mathcal{S}$. In particular, an approximate confidence measure can be $\hat{c} = \eta = |T|/|\mathcal{S}|$. This reduction is gap-preserving, since we have:

- if $|Q^*(\mathcal{G})| \geq \hat{c}|U|$, then $|T^*(\varphi, \mathcal{S})| \geq \hat{c}|\mathcal{S}|$,
- if $|Q^*(\mathcal{G})| < \frac{1}{\alpha}\hat{c}|U|$, then $|T^*(\varphi, \mathcal{S})| < \frac{1}{\alpha}\hat{c}|\mathcal{S}|$,

where $Q^*(\mathcal{G})$ denotes a maximum clique for $\mathcal{G}$, and $T^*(\varphi, \mathcal{S})$ is a maximum keeping set for dataspace $\mathcal{S}$ w.r.t. φ. The NP-hardness of approximating MAXIMUM CLIQUE to within any constant factor α has been recognized in [18]. Recall that we have the exact confidence measure $\mathsf{conf}(\varphi, \mathcal{S}) = |T^*(\varphi, \mathcal{S})|/|\mathcal{S}|$. Hence, the confidence measure is NP-hard to approximate to within in any constant factor as well. □

Despite the hardness of approximation with respect to constant factor, a heuristic greedy algorithm is known for the maximum independent set problem with performance ratio on the maximum degree in a graph [24]. We thus greedily compute approximate confidence as follows.

Specifically, we can iteratively select an object that 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 Sect. 8 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 [24].

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

Require: A dependency φ over a dataspace $\mathcal{S}$
Ensure: 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: $\quad t := \arg\min_{u \in U} |\{w \mid w \in U, (w, u) \in E\}|$
6: $\quad T := T \cup \{t\}$
7: $\quad U := U - \{t\} \cup \{w \mid w \in U, (w, t) \in E\}$
8: **return** $|T|/|\mathcal{S}|$

Proposition 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 $O(n^2)$.*

5.2 Randomized approach

Greedy 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 [33] and the confidence of approximate conditional functional dependencies [11]. Similarly, motivated by the randomized algorithm for the minimum vertex cover problem [42], 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 ϵ, $0 \leq \epsilon \leq 1$, 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, that is, Δ.

5.2.1 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 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,

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

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

we have

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

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

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

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', that is, 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 [42]. 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($\varphi, \mathcal{S}, \epsilon, \delta$)

Require: A dependency φ over a dataspace $\mathcal{S}$, additive error ϵ, and probability guarantee δ
Ensure: 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: $E := \{(t_i, t_j) \in \mathcal{T} \mid (t_i, t_j) \asymp \text{LHS}(\varphi), (t_i, t_j) \not\asymp \text{RHS}(\varphi)\}$
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 $(2 \log \Delta + 1)$ [42]. Together with the probability guarantee on sampling in Lemma 1, Proposition 3 is concluded.

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

5.2.2 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 2 *Let* $m = \frac{2}{\epsilon^2} \log \frac{2}{1-\delta}$, *we have*

$$\Pr[c' - \epsilon \le \hat{c} \le c'] \ge \delta,$$

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 1, we have

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

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

$$\Pr\left[c' - \epsilon \le \hat{c} \le c'\right] \ge \delta,$$

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 [24]. 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 $3c/(\Delta + 2) \le c' \le c$. The pseudo-code with details for estimated confidence is given in Algorithm 4.

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

Require: A dependency φ over a dataspace $\mathcal{S}$, additive error ϵ, and probability guarantee δ
Ensure: 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: $E := \{(t_i, t_j) \in \mathcal{T} \mid (t_i, t_j) \asymp \text{LHS}(\varphi), (t_i, t_j) \not\asymp \text{RHS}(\varphi)\}$
5: $U :=$ objects in $\mathcal{T}$
6: **while** $U \neq \emptyset$ **do**
7: **for** each $u \in U$ such that $d(u) \le \frac{\sum_{(w,u)\in E} d(w)}{d(u)}$ **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 [24], it is sufficient to move an object u to T if $d(u) \le \frac{\sum_{(w,u)\in E} d(w)}{d(u)}$. The computation process can be completed in $\min(\Delta^4 \log n, \Delta!)$ rounds [24]. 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. Consequently, the algorithm complexity is $O(\Delta^{\min(\Delta^4 \log n, \Delta^\Delta)} m)$. Combining the result in Lemma 2, we have

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

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

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

5.2.3 Randomized versus distributed

Recall that dataspaces may collect heterogeneous data in various sources. It is often unlike to store all the data in a centralized environment. The greedy algorithm may require high communication cost to exchange the violation results, while the randomized algorithm can naturally be deployed in the distributed environment. We can 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). Thereby, there is no exchange of violation data occurred between distributed nodes, and the cost of aggregation of measure results is low.

6 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 of data dependencies over static data. As the first step, in this paper, we also focus on the discovery in a given data set. In dataspace scenario, the major difference is that comparable correspondences are not fully identified in dataspaces. Instead, comparable correspondences are practically identified in a *pay-as-you-go* style [29]. Therefore, the discovery of dependencies should be conducted in an incremental way with respect to new identified comparison functions. 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.

6.1 Overview

When a comparison function $\theta(A_i, A_j)$ is identified, the way of their values to be compared is recognized, for example, based on equality [53] or metric distance less than certain threshold [17]. Mechanisms have been well studied in schema matching [44] and reference reconciliation [14] 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 [26,27] and FastFDs [54]. 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 [38,40]. As mentioned in Sect. 4, 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 targeting on 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[3] measure.

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, that is, redundant dependencies may exist in the returned answers.

6.2 Approximate implication

Approximate inference of functional dependencies has been considered with respect to error measure [33]. 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, for example, by augmentation.

Proposition 5 *Consider a dependency φ_1 over $\mathcal{S}$. Let*

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

It always has $g_3(\varphi_1, \mathcal{S}) \geq g_3(\varphi_2, \mathcal{S})$ 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 $\mathcal{S}$, that is, $\mathsf{conf}(\varphi_1, \mathcal{S}) = \frac{|T_1^*|}{|\mathcal{S}|}$. First, for any two objects $t_i, t_j \in T_1^*$, we have $(t_i, t_j) \models \varphi_1$ according to the definition of keeping set. It follows $(t_i, t_j) \models \varphi_2$ 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^*, that is, $T_2 \supset T_1^*$. To sum up, we have $T_2^* \supseteq T_1^*$ and the conclusion is proved according to error/confidence definition. □

6.3 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 5 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 [26,27]. 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.

6.4 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

[3] 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 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, that is, 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 5.

Algorithm 5 Incremental DISCOVERY(Σ, η, θ)

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

For example, let us consider the current function set $\Theta = \{\theta(\mathsf{manu}, \mathsf{prod}), \theta(\mathsf{website}, \mathsf{website})\}$ and $\Sigma = \emptyset$. For the new function $\theta(\mathsf{addr}, \mathsf{post})$, we consider the incremental discovery in two aspects. First, following Lines 1-5 in Algorithm 5, $\theta(\mathsf{addr}, \mathsf{post})$ is used in RHS in the dependency. Supposing $k = 3$, the LEVEL- WISE($\Theta, k-1$) function in Line 1 will return a $L = \{\theta(\mathsf{manu}, \mathsf{prod}), \theta(\mathsf{website}, \mathsf{website}), \theta(\mathsf{manu}, \mathsf{prod}) \wedge \theta(\mathsf{website}, \mathsf{website})\}$ of all function subsets (w.r.t. Θ) with length ≤ 2. Consequently, the dependency

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

in the introduction could be generated. Second, Lines 6-13 will consider the new $\theta(\mathsf{addr}, \mathsf{post})$ in LHS of a dependency. The corresponding $L = \{\theta(\mathsf{manu}, \mathsf{prod}), \theta(\mathsf{website}, \mathsf{website})\}$ is with function subset length ≤ 1. As the currently computed results have $\theta(\mathsf{manu}, \mathsf{prod}) \rightarrow \theta(\mathsf{website}, \mathsf{website}) \notin \Sigma$ yet, the dependency

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

in Example 2 could be added into Σ as well by Line 12.

7 Semantic query optimization

In this section, we study the query optimization by the proposed comparable dependencies in dataspaces, know as the *semantic query optimization* problem. To perform the optimization, we also introduce the inference rules for implicating CDs.

7.1 Preliminary

Integrity constraints (e.g., FDs) can be utilized to rewrite a query in order to optimize the query evaluation. It is known as the semantic query optimization problem [36]. In the dataspaces scenario, the major difference is about comparison functions. A query over dataspaces not only considers the attributes specified in the query but also expands to other attributes according to comparison correspondences [13].

We consider a dataspace $\mathcal{S}$ with a set Θ of all comparison functions. A *query object*,

$$q = \{(A_1 : v_1), \ldots, (A_{|Q|} : v_{|Q|})\},$$

specifies predicates on a set of attributes $Q = \{A_1, \ldots, A_{|Q|}\}$. It is to return all the objects t in $\mathcal{S}$ with comparable attribute values to q, that is, for each $A_i \in Q$ of q, we can find a B_i of t, such that $(q[A_i], t[B_i]) \asymp \theta(A_i, B_i)$, where $\theta(A_i, B_i) \in \Theta$. Obviously, there may exist multiple $\theta(A_i, B_i)$ associated to an attribute A_i. The query needs that at least one of them is true. In other words, we want to find all the objects t such that

$$(q, t) \asymp \bigwedge_{A_i \in Q} \Big(\bigvee_{B_i \text{ of } t} \theta(A_i, B_i) \Big), \theta(A_i, B_i) \in \Theta.$$

Let $\Theta(A_i) = \{\theta(A_i, B_i) \in \Theta \mid B_i \in \mathsf{attr}(\mathcal{S})\}$ be the set of all comparison functions in Θ that are associated on attribute A_i. Let

$$\Phi[Q] = \Theta(A_1) \times \cdots \times \Theta(A_{|Q|})$$

where each element $\phi[Q] \in \Phi[Q]$ is a pattern of comparison functions w.r.t. attributes in Q, such as

$$\phi[Q] = \theta(A_1, B_1) \wedge \cdots \wedge \theta(A_{|Q|}, B_{|Q|}) \in \Phi[Q]$$

where $\theta(A_i, B_i) \in \Theta(A_i)$. The query requires that at least one $\phi[Q]$ is true and can be equivalently represented by

$$(q, t) \asymp \bigvee_{\phi[Q] \in \Phi[Q]} \phi[Q].$$

Example 4 Given a dataspace $\mathcal{S}$ in Example 1, let

$$\Theta = \{\theta(\mathsf{manu}, \mathsf{prod}), \theta(\mathsf{post}, \mathsf{addr}), \theta(\mathsf{post}, \mathsf{post}), \ldots\}$$

be the set of comparison functions. We consider a query object

$$q : \{(\mathsf{manu} : \mathsf{Apple}), (\mathsf{post} : \mathsf{InfiniteLoop}, \mathsf{CA})\}.$$

According to the dataspace query processing, we need to consider all the attributes that are comparable to the query attributes, having

$$\Theta(\mathsf{manu}) = \{\theta(\mathsf{manu}, \mathsf{manu}), \theta(\mathsf{manu}, \mathsf{prod})\},$$
$$\Theta(\mathsf{post}) = \{\theta(\mathsf{post}, \mathsf{post}), \theta(\mathsf{post}, \mathsf{addr})\}.$$

Consequently, the query returns the objects that agree on at least one of the following conditions in $\Phi[Q]$,

$$\phi_1[Q] = \theta(\mathsf{manu}, \mathsf{manu}) \wedge \theta(\mathsf{post}, \mathsf{post}),$$
$$\phi_2[Q] = \theta(\mathsf{manu}, \mathsf{manu}) \wedge \theta(\mathsf{post}, \mathsf{addr}),$$
$$\phi_3[Q] = \theta(\mathsf{manu}, \mathsf{prod}) \wedge \theta(\mathsf{post}, \mathsf{post}),$$
$$\phi_4[Q] = \theta(\mathsf{manu}, \mathsf{prod}) \wedge \theta(\mathsf{post}, \mathsf{addr}),$$

that is, $(q, t) \asymp \phi_1[Q] \vee \cdots \vee \phi_4[Q]$. It will find all the three objects in $\mathcal{S}$, since they are comparable with the query object q on (manu, prod) and (addr, post), respectively.

7.2 Implication rules

In order to optimize the queries by data dependencies, we need to study the implication of CDs. The following inference rules are presented for exact dependencies. As we will see soon at the end of this section, the query optimization ensures the completeness of the answer set, that is, without losing any query result, even when approximate dependencies are applied.

To investigate the implication of dependencies in dataspaces, we first adapt Armstrong's inference rules [2] with comparison functions to dataspaces as follows. Specifically, given any dataspace $\mathcal{S}$ and a dependency φ, we always have:

A1. (reflexivity) $\mathcal{S} \vDash \text{LHS}(\varphi) \wedge \theta(A_i, A_j) \rightarrow \theta(A_i, A_j)$.
A2. (augmentation) If $\mathcal{S} \vDash \varphi$, that is, $\mathcal{S} \vDash \text{LHS}(\varphi) \rightarrow \text{RHS}(\varphi)$, then $\mathcal{S} \vDash \text{LHS}(\varphi) \wedge \theta(A_i, A_j) \rightarrow \text{RHS}(\varphi) \wedge \theta(A_i, A_j)$.
A3. (transitivity) If $\mathcal{S} \vDash \varphi$ and $\mathcal{S} \vDash \text{RHS}(\varphi) \rightarrow \theta(A_i, A_j)$, then $\mathcal{S} \vDash \text{RHS}(\varphi) \rightarrow \theta(A_i, A_j)$.

As these rules are adapted from the Armstrong axioms, the semantics and soundness self-explained. For instance, the reflexivity rule A1 states that any two objects agreeing $\text{LHS}(\varphi) \wedge \theta(A_i, A_j)$ will always agree $\theta(A_i, A_j)$.

Besides the above three rules, we may also investigate the implication particular to comparable dependencies, for example, with respect to comparison function as follows:

A4. (containment) $\mathcal{S} \vDash \theta(A_i, A_i) \rightarrow \theta(A_i, A_j)$.

The soundness of A4 is justified as follows. Recall that $\theta(A_i, A_j)$ requires at least one of the comparison operators to be true, $A_i \leftrightarrow_{ii} A_i$, $A_i \leftrightarrow_{ij} A_j$ or $A_j \leftrightarrow_{jj} A_j$. For the comparison function $\theta(A_i, A_i)$ with aligned attributes, it already implies $A_i \leftrightarrow_{ii} A_i$ to be true. Therefore, two objects agreeing $\theta(A_i, A_i)$ should always agree $\theta(A_i, A_j)$ as well.

The aforesaid four rules might not be a complete and sound inference system for implying comparable dependencies. As discussed at the end of Sect. 9, relationships among comparison operators may also be employed in the implication. Indeed, it is highly non-trivial to study inference systems when metric operator exists [48]. We focus on part of the inference rules applicable to the semantic query optimization below and leave the attempt of developing a sound and complete set of inference rules in the future work.

7.3 Predicate elimination

Predicate elimination is a typical technique in query rewriting with data dependencies. The basic idea is: if a predicate is known to be always true, then it can be eliminated from query. Since the rewritten query has less predicates to evaluate, query performance can be improved. Following the same line, we mainly study two opportunities for eliminating predicates in dataspace queries.

First, suppose that there is a dependency $\phi_i[Q] \rightarrow \phi_j[Q]$ where $\phi_i[Q], \phi_j[Q]$ are two patterns in $\Phi[Q]$. Then, the query can be rewritten by

$$(q, t) \asymp \bigvee_{\phi[Q] \in \Phi[Q] \setminus \{\phi_i[Q]\}} \phi[Q]. \tag{3}$$

That is, we eliminate $\phi_i[Q]$ from the query condition $\Phi[Q]$. According to the dependency, any objects agreeing $\phi_i[Q]$ should also agree $\phi_j[Q]$. Consequently, the set of answers having $(q, t) \asymp \phi_i[Q]$ should be a subset of the answers $(q, t) \asymp \phi_j[Q]$. Referring to the OR relationship between patterns, it is safe to rewrite the query by removing $\phi_i[Q]$ from $\Phi[Q]$.

Second, suppose that there is a dependency $\phi[X] \rightarrow \phi[Y]$ where $\phi[X], \phi[Y]$ are projections of $\phi[Q]$ on attributes $X \subseteq Q, Y \subseteq Q$. According to the semantics of dependencies, the query can be rewritten by:

$$(q, t) \asymp \bigvee_{\phi[Q] \in \Phi[Q]} \phi[Q \setminus Y]. \tag{4}$$

In other words, $\phi[Y]$ is eliminated from the query predicates. Again, as analyzed before, the answer set having $(q, t) \asymp \phi[X]$ should be a subset of $(q, t) \asymp \phi[Y]$. Thereby, the condition $\phi[X] \wedge \phi[Y]$ is essentially equivalent to $\phi[X]$. We can safely eliminate $\phi[Y]$ from the predicates too.

Since less comparison functions need to be considered, the query efficiency is improved by applying the aforesaid two eliminations.

Example 5 We use the query in Example 4 to illustrate the predicate elimination.

First, let's consider the elimination among $\phi_i[Q]$ in $\Phi[Q]$. According to the containment rule A4, we have the following dependency

$$\theta(\mathsf{post}, \mathsf{post}) \rightarrow \theta(\mathsf{post}, \mathsf{addr}).$$

By applying the augmentation rule A2, we can further expend the rule as

$$\theta(\mathsf{manu}, \mathsf{manu}) \wedge \theta(\mathsf{post}, \mathsf{post}) \rightarrow \theta(\mathsf{manu}, \mathsf{manu}) \wedge \theta(\mathsf{post}, \mathsf{addr}),$$

that is, $\phi_1[Q] \rightarrow \phi_2[Q]$. Referring to the first elimination type in formula (3), we can safely ignore $\phi_1[Q]$ in the query. Similarly, $\phi_2[Q]$ and $\phi_3[Q]$ can also be eliminated according to $\phi_2[Q] \rightarrow \phi_4[Q]$ and $\phi_3[Q] \rightarrow \phi_4[Q]$, respectively. The query is eventually rewritten as $(q, t) \asymp \phi_4[Q]$ after these safe predicate elimination steps.

Next, we further investigate the elimination inside a pattern $\phi[Q]$. Suppose that we have a dependency

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

According to the second type of elimination in formula (4), the query evaluation with respect to $\phi_4[Q]$, that is, $(q, t) \asymp \theta(\mathsf{manu}, \mathsf{prod}) \wedge \theta(\mathsf{post}, \mathsf{addr})$, can be rewritten by $(q, t) \asymp \theta(\mathsf{manu}, \mathsf{prod})$.

It is notable that the query is equivalently rewritten if exact dependencies (with 0.0 error or 1.0 confidence) are applied. Query answers are exactly the same as those without using such dependencies. When approximate dependencies are adopted, the returned answers will always be a super set of exact ones since we rewrite the query by eliminating part of the original query conditions. In other words, applying approximate dependencies will never lose answers. Such conclusion is also observed in our experiments where the *recall* accuracy of the query answers is always equal to 1.0. Therefore, in the experiments in Sect. 8.3, we mainly observe the *precision* accuracy of returned results. The experimental results also verify that the predicate elimination can significantly reduce the query time cost.

8 Experiments

In this section, we report an extensive experimental evaluation of proposed mechanisms on two real data sets, Base and Wiki. Google Base[4] 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 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.

Another real data set of dataspaces is from Wikipedia,[5] 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 3 Example of dependencies

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

Table 3 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, that is, the manufacture product number and unified product code, respectively. The dependency φ_1 in Table 3 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,[6] our experiments in Sect. 8.1 evaluate both the effectiveness and efficiency of various computation algorithms for error and confidence measures. Moreover, in Sect. 8.2, we present the performance of pay-as-you-go discovery. Finally, Sect. 8.3 reports the effectiveness of applying data dependencies in query optimization. We adopt the widely used cosine similarity with q-grams [41] as the comparison operator. Similar results in the secondary Wiki data set may be omitted. All algorithms are implemented by Java. Experiments run on a machine with Intel Core 2 CPU (2.13 GHz) and 2 GB of memory.

8.1 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 Figs. 1a and 2a, 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

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

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

[6] e.g., in Table 3 discovered by our Algorithm 5.

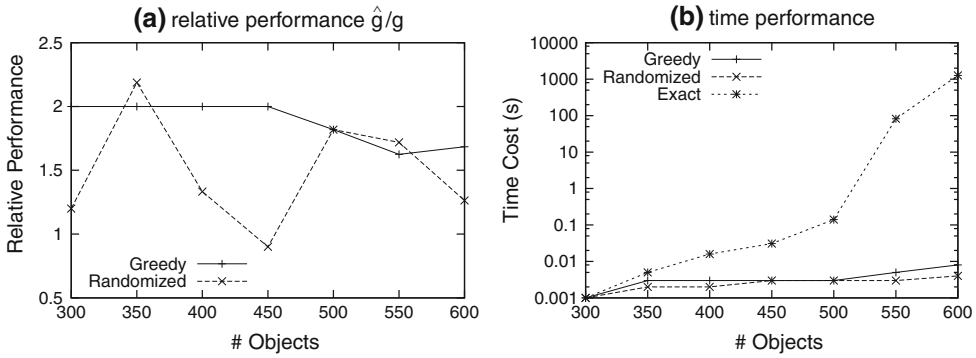

Fig. 1 Performance ($\hat{g}/g$, time) of error approximation

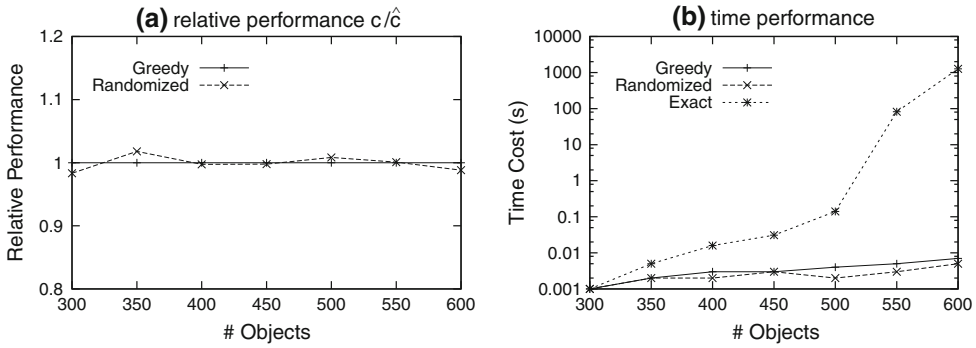

Fig. 2 Performance ($c/\hat{c}$, time) of confidence approximation

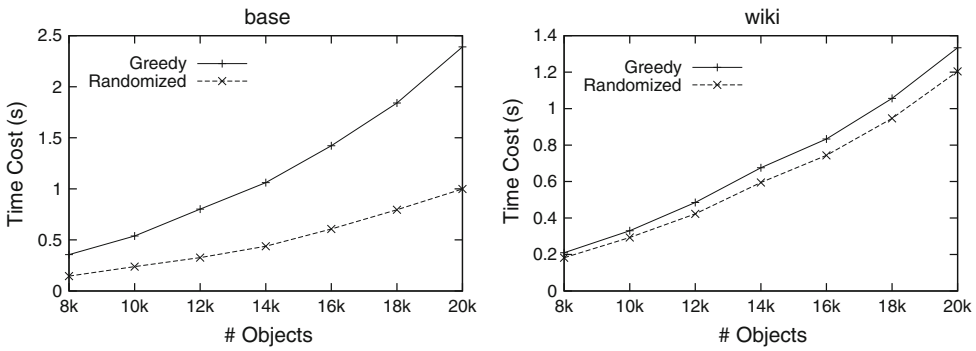

Fig. 3 Scalability of error approximation

almost hold, with low error/high confidence. For example, the confidence of φ_3 in Table 3 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 performance of error measure, for example, 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, for example, under 350 objects in Fig. 1a, it is bounded with high probability (guaranteed by $\delta = 0.8$).

Obviously, exact computation of validation does not scale well. As presented in Figs. 1b and 2b, 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, Figs. 3 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 the corresponding effects on time costs. Generally, the larger the additive ϵ is allowed, the less the number of samples are 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 Fig. 5b, c are roughly affected by the size of samples.

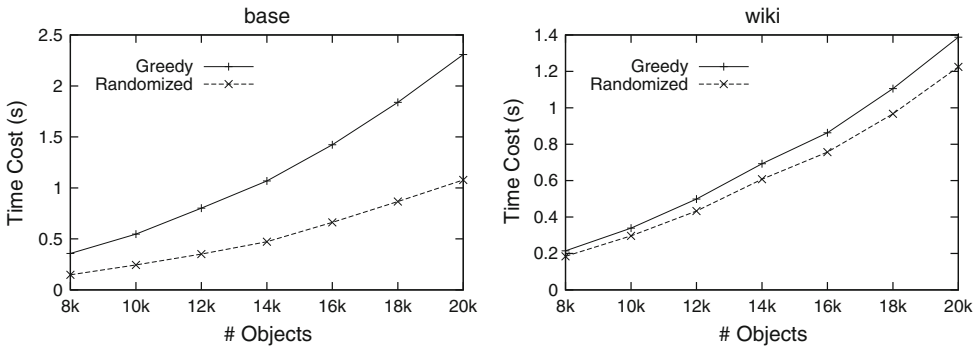

Fig. 4 Scalability of confidence approximation

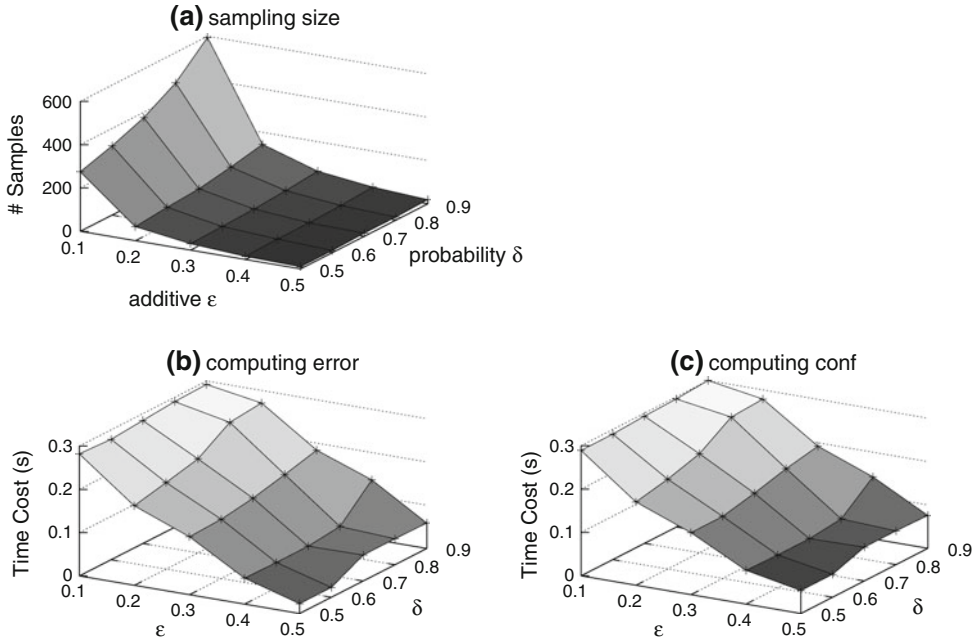

Fig. 5 Sampling size and time performance of randomized algorithm with various additive ϵ and probability δ

8.2 Discovery evaluation

In this experiment set, we evaluate the pay-as-you-go discovery of dependencies by calling Algorithm 5 incrementally, that is, Σ :=DISCOVERY(Σ, η, θ). The discovery experiments also use 10k objects in the data set. Given a fixed k, the result sizes are very close with a small variance under different numbers of objects. When $k = 4$, there are about 232653 CDs returned. It reduce to 26517 when $k = 3$. There is only 3639 for $k = 2$. We mainly observe the time cost under these various settings in the following.

Figure 6a illustrates the incremental discovery of dependencies with the increase in 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 5. As presented in Fig. 6b with various error measure requirements η, a larger error requirement η needs less time cost.

As shown in Fig. 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 Sect. 5 for validating dependencies.

8.3 Query optimization evaluation

This set of experiments evaluate the performance of query optimization by comparable dependencies as illustrated in Sect. 7. The query used in experiments has three predicates, including attributes mpn, id and imagelink. We follow the convention of evaluating semantic query optimization [7,25,52] as discussed in Sect. 2, that is, to observe the query performance with and without query rewriting by data dependencies.

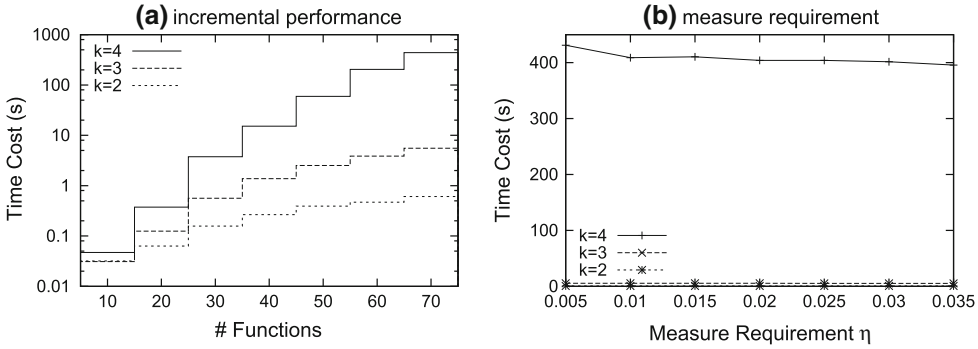

Fig. 6 Performance of pay-as-you-go discovery

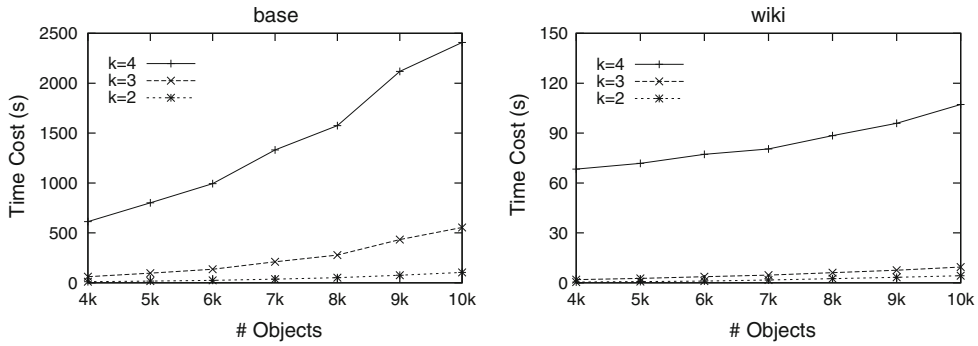

Fig. 7 Scalability of pay-as-you-go discovery

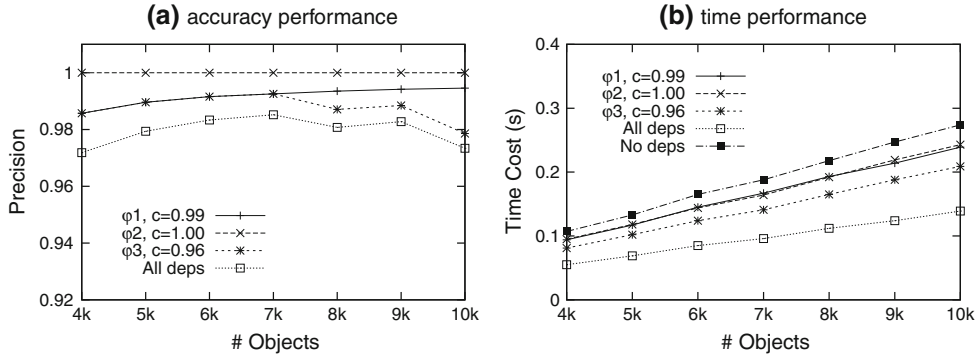

Fig. 8 Performance of query optimization

To demonstrate the query accuracy, we observe the precision of answers. As stated, the query with (approximate) dependencies will never lose answers. That is, the recall of query answers will always be 1.0, which is also observed in our experiments and not reported. Figure 8 first illustrates the results by applying three different dependencies in Table 3 individually, for example, $\varphi_1, c = 0.99$ denotes the dependency φ_1 with confidence 0.99. Moreover, results with all three dependencies and with no dependencies are also presented.

By applying exact data dependencies, the query answer is equivalent to the one without applying dependencies. Therefore, as presented in Fig. 8a, the precision of φ_2 with confidence $c = 1.00$ is always 1. A dependency with lower confidence may return more false answers, that is, lower query precision by φ_3 with confidence $c = 0.96$. When several approximate dependencies are applied together, false answers of each one appear together in the results and thus the accuracy drops.

On the other hand, as shown in Fig. 8b, the query efficiency is improved when dependencies are applied. It is natural that the more the dependencies could be applied, the lower the query costs would be. By considering all dependencies, the query efficiency is significantly improved compared with the no dependency query. Meanwhile, there is no much accuracy lost, with precision about 0.98 under most tests as presented in Fig. 8a.

Finally, to observe the performance of applying low quality dependencies, we introduce an artificial CD φ_4 with low confidence 0.41.

$$\varphi_4 : \theta(\mathsf{id}, \mathsf{id}) \rightarrow \theta(\mathsf{imagelink}, \mathsf{imagelink})$$

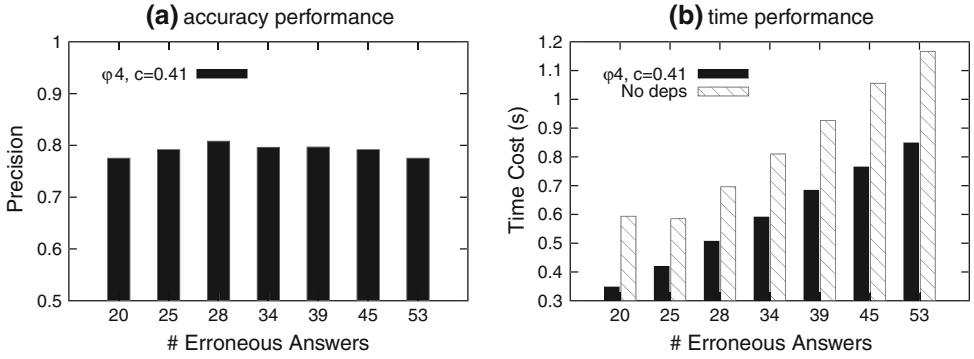

Fig. 9 Performance of query optimization under low quality dependencies

As shown in aforesaid results, there will be more erroneous answers when applying such low confidence dependencies. Therefore, Fig. 9 mainly reports the performance under various erroneous answers in different tests. As illustrated, the erroneous answers increase from 20 to 53, in the 7 tests with data sizes ranging from 4k to 10k. Although the precision accuracy in Fig. 9a is low due to the weak confidence dependency, the precision results are quite stable in various erroneous answer sizes. The results demonstrate the robustness of applying CDs in query optimization. The time cost Fig. 9b grows linearly again as it depends on the data sizes, which is increasing linearly from 4k to 10k in this experiment.

9 Conclusions and discussions

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 dataspaces with the consideration of comparable correspondences. Unfortunately, due to heterogeneous data, as we proved, it is already hard to tell 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. The semantic query optimization with comparable dependencies in dataspaces is also illustrated, together with an extensive experimental evaluation on real data sets.

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. Besides the implication rules introduced in Sect. 7.2, further inference rules can be introduced when the relationships among comparison functions are identified. For example, an equality operator '=' can always be interpreted as a special case of metric operator '$\approx_{=0}$', that is, with metric distance = 0. Moreover, if more than one comparison operators are associated to an attribute pair, any two objects in a dataspace $\mathcal{S}$ agreeing on a comparison function with equality must always agree on the corresponding comparison function with metric operator, for example, '$\approx_{\leq 7}$'. Consequently, more types of inference rules can be introduced, which is out the scope of this study. We leave the interesting topic as future work to develop a sound and complete inference system under certain premise of comparison functions.

Acknowledgments This work is supported in part by the Hong Kong RGC GRF Project No.611411, National Grand Fundamental Research 973 Program of China under Grant 2012-CB316200, HP IRP Project 2011, Microsoft Research Asia Grant, MRA11EG05 and HKUST RPC Grant RPC10EG13, and US NSF through grants IIS-0905215,IIS-0914934, DBI-0960443, OISE-0968341, OIA-0963278.

References

1. Arenas, M., Bertossi, L.E., Chomicki, J.: Consistent query answers in inconsistent databases. In: PODS, pp. 68–79 (1999)
2. Armstrong, W.W.: Dependency structures of data base relationships. In: IFIP Congress, pp. 580–583 (1974)
3. Bertossi, L.E., Bravo, L., Franconi, E., Lopatenko, A.: The complexity and approximation of fixing numerical attributes in databases under integrity constraints. Inf. Syst. **33**(4-5), 407–434 (2008)
4. Bitton, D., Millman, J., Torgersen, S.: A feasibility and performance study of dependency inference. In: ICDE, pp. 635–641 (1989)
5. Bohannon, P., Fan, W., Geerts, F., Jia, X., Kementsietsidis, A.: Conditional functional dependencies for data cleaning. In: ICDE, pp. 746–755 (2007)
6. Chakravarthy, U.S., Grant, J., Minker, J.: Logic-based approach to semantic query optimization. ACM Trans. Database Syst. **15**(2), 162–207 (1990)
7. Cheng, Q., Gryz, J., Koo, F., Leung, T.Y.C., Liu, L., Qian, X., Schiefer, K.B.: Implementation of two semantic query optimization techniques in db2 universal database. In: VLDB, pp. 687–698 (1999)
8. Chiang, F., Miller, R.J.: Discovering data quality rules. PVLDB **1**(1), 1166–1177 (2008)
9. Chomicki, J.: Semantic optimization techniques for preference queries. Inf. Syst. **32**(5), 670–684 (2007)
10. Chomicki, J., Marcinkowski, J.: Minimal-change integrity maintenance using tuple deletions. Inf. Comput. **197**(1-2), 90–121 (2005)

11. Cormode, G., Golab, L., Korn, F., McGregor, A., Srivastava, D., Zhang, X.: Estimating the confidence of conditional functional dependencies. In: SIGMOD Conference, pp. 469–482 (2009)
12. Dinur, I., Safra, S.: The importance of being biased. In: STOC, pp. 33–42 (2002)
13. Dong, X., Halevy, A.Y.: Indexing dataspaces. In: SIGMOD Conference, pp. 43–54 (2007)
14. Elmagarmid, A.K., Ipeirotis, P.G., Verykios, V.S.: Duplicate record detection: a survey. IEEE Trans. Knowl. Data Eng. **19**(1), 1–16 (2007)
15. Fan, W.: Dependencies revisited for improving data quality. In: PODS, pp. 159–170 (2008)
16. Fan, W., Geerts, F., Lakshmanan, L.V.S., Xiong, M.: Discovering conditional functional dependencies. In: ICDE, pp. 1231–1234 (2009)
17. Fan, W., Li, J., Jia, X., Ma, S.: Reasoning about record matching rules. In: PVLDB (2009)
18. Feige, U., Goldwasser, S., Lovász, L., Safra, S., Szegedy, M.: Approximating clique is almost np-complete (preliminary version). In: FOCS, pp. 2–12 (1991)
19. Flach, P.A., Savnik, I.: Database dependency discovery: a machine learning approach. AI Commun. **12**(3), 139–160 (1999)
20. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman, London (1979)
21. Giannella, C., Robertson, E.L.: On approximation measures for functional dependencies. Inf. Syst. **29**(6), 483–507 (2004)
22. Golab, L., Karloff, H.J., Korn, F., Srivastava, D., Yu, B.: On generating near-optimal tableaux for conditional functional dependencies. PVLDB **1**(1), 376–390 (2008)
23. Halevy, A.Y., Franklin, M.J., Maier, D.: Principles of dataspace systems. In: PODS, pp. 1–9 (2006)
24. Halldórsson, M.M., Radhakrishnan, J.: Greed is good: approximating independent sets in sparse and bounded-degree graphs. In: STOC, pp. 439–448 (1994)
25. Hsu, C.N., Knoblock, C.A.: Semantic query optimization for query plans of heterogeneous multidatabase systems. IEEE Trans. Knowl. Data Eng. **12**(6), 959–978 (2000)
26. Huhtala, Y., Kärkkäinen, J., Porkka, P., Toivonen, H.: Efficient discovery of functional and approximate dependencies using partitions. In: ICDE, pp. 392–401 (1998)
27. Huhtala, Y., Kärkkäinen, J., Porkka, P., Toivonen, H.: Tane: an efficient algorithm for discovering functional and approximate dependencies. Comput. J. **42**(2), 100–111 (1999)
28. Ilyas, I.F., Markl, V., Haas, P.J., Brown, P., Aboulnaga, A.: Cords: automatic discovery of correlations and soft functional dependencies. In: SIGMOD Conference, pp. 647–658 (2004)
29. Jeffery, S.R., Franklin, M.J., Halevy, A.Y.: Pay-as-you-go user feedback for dataspace systems. In: SIGMOD Conference, pp. 847–860 (2008)
30. Karakostas, G.: A better approximation ratio for the vertex cover problem. ACM Trans. Algorithm. **5**(4), 1–8 (2009). doi:10.1145/1597036.1597045
31. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (eds.) Complexity of Computer Computations, Plenum Press, Berlin, pp. 85–103 (1972)
32. King, R.S., Legendre, J.J.: Discovery of functional and approximate functional dependencies in relational databases. JAMDS **7**(1), 49–59 (2003)
33. Kivinen, J., Mannila, H.: Approximate inference of functional dependencies from relations. Theor. Comput. Sci. **149**(1), 129–149 (1995)
34. Koudas, N., Saha, A., Srivastava, D., Venkatasubramanian, S.: Metric functional dependencies. In: ICDE, pp. 1275–1278 (2009)
35. Kramer, S., Pfahringer, B.: Efficient search for strong partial determinations. In: KDD, pp. 371–374 (1996)
36. Levy, A.Y., Sagiv, Y.: Semantic query optimization in datalog programs. In: PODS, pp. 163–173 (1995)
37. Madhavan, J., Cohen, S., Dong, X.L., Halevy, A.Y., Jeffery, S.R., Ko, D., Yu, C.: Web-scale data integration: you can afford to pay as you go. In: CIDR, pp. 342–350 (2007)
38. Mannila, H., Räihä, K.J.: Dependency inference. In: VLDB, pp. 155–158 (1987)
39. Mannila, H., Räihä, K.J.: Design of Relational Databases. Addison-Wesley, Boston (1992)
40. Mannila, H., Räihä, K.J.: Algorithms for inferring functional dependencies from relations. Data Knowl. Eng. **12**(1), 83–99 (1994)
41. Navarro, G.: A guided tour to approximate string matching. ACM Comput. Surv. **33**(1), 31–88 (2001)
42. Parnas, M., Ron, D.: Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. Theor. Comput. Sci. **381**(1-3), 183–196 (2007)
43. Pfahringer, B., Kramer, S.: Compression-based evaluation of partial determinations. In: KDD, pp. 234–239 (1995)
44. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. VLDB J. **10**(4), 334–350 (2001)
45. Salles, M.A.V., Dittrich, J., Blunschi, L.: Intensional associations in dataspaces. In: ICDE (2010)
46. Salles, M.A.V., Dittrich, J.P., Karakashian, S.K., Girard, O.R., Blunschi, L.: Itrails: pay-as-you-go information integration in dataspaces. In: VLDB, pp. 663–674 (2007)
47. Sarma, A.D., Dong, X., Halevy, A.Y.: Bootstrapping pay-as-you-go data integration systems. In: SIGMOD Conference, pp. 861–874 (2008)
48. Song, S., Chen, L.: Differential dependencies: reasoning and discovery. ACM Trans. Database Syst. **36**(3), 16 (2011)
49. Song, S., Chen, L., Cheng, H.: Parameter-free determination of distance thresholds for metric distance constraints. In: ICDE (2012, to appear)
50. Song, S., Chen, L., Yu, P.S.: On data dependencies in dataspaces. In: ICDE, pp. 470–481 (2011)
51. Song, S., Chen, L., Yuan, M.: Materialization and decomposition of dataspaces for efficient search. IEEE Trans. Knowl. Data Eng. **23**(12), 1872–1887 (2011)
52. Su, H., Rundensteiner, E.A., Mani, M.: Semantic query optimization for xquery over xml streams. In: VLDB, pp. 277–288 (2005)
53. Wang, D.Z., Dong, X.L., Sarma, A.D., Franklin, M.J., Halevy, A.Y.: Functional dependency generation and applications in pay-as-you-go data integration systems. In: WebDB (2009)
54. Wyss, C.M., Giannella, C., Robertson, E.L.: Fastfds: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances-extended abstract. In: DaWaK, pp. 101–110 (2001)