Metric Functional Dependencies

Nick Koudas [*1], Avishek Saha [#2], Divesh Srivastava [†3], Suresh Venkatasubramanian [#4]

[*]*Department of Computer Science, University of Toronto, Toronto ON M5S 2E4, Canada*
[1]koudas@cs.toronto.edu

[#]*School Of Computing, University of Utah, Salt Lake City UT 84112, USA*
[2]avishek@cs.utah.edu, [4]suresh@cs.utah.edu

[†]*AT&T Labs-Research, Florham Park, NJ 07932, USA*
[3]divesh@research.att.com

***Abstract*—When merging data from various sources, it is often the case that small variations in data format and interpretation cause traditional functional dependencies (FDs) to be violated, without there being an intrinsic violation of semantics. Examples include differing address formats, or different reported latitude/longitudes for a given address. In this paper, we define metric functional dependencies, which strictly generalize traditional FDs by allowing small differences (controlled by a metric) in values of the consequent attribute of an FD. We present efficient algorithms for the verification problem: determining whether a given metric FD holds for a given relation. We experimentally demonstrate the validity and efficiency of our approach on various data sets that lie in multidimensional spaces.**

I. Introduction

Functional dependencies (FDs) are fundamental constraints that define the relation between attributes in a database. Key relationships are special kinds of functional dependencies, and FDs provide a mechanism for database normalization during the design process. In its most basic form, a functional dependency over a relation instance r of a relation schema R is an expression $X \rightarrow Y$, where $X, Y \subseteq R$. R is said to satisfy the dependency $X \rightarrow Y$ if all tuples with any fixed value of the attributes X share a single value of the attributes Y. This captures the idea that the values in Y *depend* on the values in X.

However, this formulation of a dependency is not robust enough to capture functional relationships on data obtained from merging heterogeneous sources, each having possibly different representation conventions for the attributes Y. Consider Table I, an example of a table of movie titles and running times obtained from crawling various web sources.

Source	Title	Duration
movies.aol.com	Aliens	110
finnguide.fi	Aliens	112
amazon.com	Clockwork Orange	137
movie-vault.com	A Beautiful Mind	144
walmart.com	A Beautiful Mind	145
tesco.com	Clockwork Orange	131

TABLE I
Discrepancies in movie durations

In a table such as this, we would expect the FD Title → Duration to hold. However, it is likely that different sources have different methods for measuring movie durations, depending on whether opening blurbs or closing credits are included, and so on. For example, we have differing values (110 and 112) in the Duration field for the first two entries. In such a setting, where there can be natural differences in how data is recorded, a traditional dependency definition seems overly restrictive.

As another example, consider Table II, which depicts an example of violation of traditional FDs for strings. Whether or not the word "Avenue" is abbreviated to "Ave.", whether "Street" is written out in full or as "St.", and so on, might change the representation of an address to the point where a natural dependency of the form SSN → Address may no longer hold on a merged table, even though the different sources are clearly referring to the same address.

Source	SSN	Address
data.com	124-14-5903	1403 3rd Avenue, Cleveland OH
data.com	563-82-5145	1701 New York Av., Washington, D.C.
snoop.com	563-82-5145	1701 New York Ave., Washington, D.C.
snoop.com	124-14-5903	1403 Third Ave., Cleveland OH

TABLE II
Example of address discrepancies

An example of violation of traditional dependencies due to variations that form a natural part of the data is depicted in Table III. The data is organized according to address and their latitude/longitude values returned from multiple geo-coding websites [1]. In this scenario one might expect a functional dependency of the form Address → (Latitude, Longitude) to hold, but small errors in precision (insignificant in general), might causes discrepancies that violate this dependency. Note that in this setting, the variations are a natural part of the data, and cannot be eliminated by format standardization.

Source	Address	Latitude	Longitude
google	65 N St Apt#C6, SLC	40.770896	-111.864066
geocoder	65 N St Apt#C6, SLC	40.770767	-111.863768
google	50 Cen Camp Dr, SLC	40.758951	-111.845246
geocoder	50 Cen Camp Dr, SLC	40.767599	-111.843995
google	35 S 700 E Apt#3, SLC	40.76837	-111.87064
geocoder	35 S 700 E Apt#3, SLC	40.76833	-111.870869

TABLE III
Discrepancies in latitude/longitude for identical addresses

There has been extensive research on the problem of ver-

ifying and enforcing integrity constraints in heterogeneous databases [2] integrated from multiple sources. Many different measures characterizing *approximate* functional dependencies have been proposed. These include measures based on deleting as few tuples as possible so that a given FD holds [3], and measures based on computing the *conditional entropy*. The paper by Gianella [4] surveys the various measures and places them within an axiomatic framework. Recently, Bohannon *et al.* [5] introduced the notion of a *conditional functional dependency*. This is a dependency that applies only to tuples that satisfy a certain condition, and is motivated by integration scenarios where different tables, with possibly differing integrity constraints, might be integrated. This work was further developed by Cong *et al.* [6], who studied the problem of determining a minimal *repair* of an inconsistent database so as to satisfy a given set of conditional FDs. Most recently, Fan [7] introduced the notion of *matching dependencies* as a way of enforcing object identification constraints across tables. These dependencies use the idea of similarity measures to generalize the equality relation. Another perspective on dependencies and data cleaning was provided by Arenas *et al.* [8]: in their framework, rather than removing tuples that cause inconsistencies with respect to dependencies, they examine queries to the database, and return an answer that would be consistent across all *minimal repairs* to the database. Research on computing such *consistent query answers* was developed further in [9], [10].

Despite the enhanced definitions, it is to be noted that all the above approaches are vulnerable to small discrepanies in data. Both approximate as well as conditional functional dependencies might give poor results due to this inherent lack of robustness in their defintions. We intend to overcome these limitations and propose a new notion of functional dependencies that capture these small variations in data. In this paper we initiate a study of metric functional dependencies. We define the notion of a metric functional dependency (MFD) and perform a study of the complexity of verifying whether a metric functional dependency holds between two attribute sets. We give exact algorithms for verifying metric FDs. Specifically, we show how to verify MFDs efficiently for general metrics as well as Euclidean distance spaces. We present an implementation of schemes for verifying MFDs, and accompany this with an experimental suite that demonstrates the ability of MFDs to extract useful structure from data in an efficient manner.

We start by defining metric FDs in Section II. In Section III we study the complexity of verifying MFDs, and give exact algorithms. Section IV presents a detailed experimental evaluation of our approach; this is followed by discussion in Section V.

II. Metric Functional Dependencies

We denote the domain of an attribute X by dom(X). If X consists of a sequence of attributes $X = A_1 A_2 \ldots A_k$, then dom(X) = dom(A_1) $\times$ dom(A_2) $\times \ldots \times$ dom(A_k). A *functional dependency* over a relation schema R is an expression $X \rightarrow Y$, where $X, Y \subseteq R$. A functional dependency is said to hold over a relation instance r if for all pairs of tuples $t, t' \in r$, $t[X] = t'[X]$ implies $t[Y] = t'[Y]$. For a tuple t and attributes X, let the equivalence class of t, denoted by $[t]_X(r)$, be the set of all tuples in r that agree with t on X. In what follows, we will drop references to r, and unless otherwise specified, $[t]_X$ will always be computed over all of r. In other words, $[t]_X = \{u \in r | u[X] = t[X]\}$. We denote the partition of r with respect to X as the collection of equivalence classes $\pi_X(r) = \{[t]_X\}$. The expression Part(r) denotes the space of all partitions of the tuples in r. In particular, $\pi_X(r) \in$ Part(r).

For a set of tuples $T \subseteq r$, let $T[Y] = \{t[Y] | t \in T\}$ denote the projection of T onto the attribute set Y. We can now write the condition for $X \rightarrow Y$ to be a functional dependency as: $\max_{T \in \pi_X(r)} |T[Y]| \leq 1$.

A. Rationale

A traditional FD $X \rightarrow Y$ says that for any two tuples t, t', $t[X] = t'[X] \Rightarrow t[Y] = t'[Y]$. In the aforementioned examples, this notion breaks down because we have a situation where $t[Y]$ and $t'[Y]$, while close in a metric sense, are not equal. Thus, in order to formulate a more robust definition of a functional dependency, we have to incorporate this notion of "closeness" into the definition. More formally, we can exploit any available *metric structure* defined on the attribute Y and replace the condition $t[Y] = t'[Y]$ by the metric condition $d(t[Y], t'[Y]) \leq \delta$, where δ is a tolerance parameter.

Consider the movie database example from Table I. A natural metric on running times is the absolute difference $d(t, t') = |t_{\text{DURATION}} - t'_{\text{DURATION}}|$. If we fix $\delta = 6$, then we see that for each of the distinct values of the attribute TITLE, the set of values of DURATION lie within an interval of length $\delta = 6$, and so a "metric" functional dependency holds. For the example involving latitude/longitude, an appropriate distance metric in a plane is the two-dimensional Euclidean metric. For the example involving addresses, the natural metric might be the distance induced by cosine similarity on strings. In general, we can say that the MFD $X \xrightarrow{\delta} Y$, with metric d : dom(Y) $\times$ dom(Y) $\rightarrow \mathbb{R}$ holds if for each $x \in$ dom(X), the set of tuples t with $t[X] = x$, when projected to Y, lie within a ball of diameter δ.

Notice that this definition strictly generalizes the traditional definition of a functional dependency. We can define the *exact metric* $d_E(y, y')$ as 1 if $y \neq y'$, and 0 if $y = y'$. It can easily be verified that this satisfies the properties of a metric. Now, setting $\delta = 0$, we can see that if $X \xrightarrow{\delta} Y$ with the metric d_E, then the standard FD $X \rightarrow Y$ holds.

B. Definitions

Let d : dom(Y) $\times$ dom(Y) $\rightarrow \mathbb{R}$ be a metric defined on the domain of Y. Specifically, d is symmetric, and satisfies the triangle inequality as well as identity of indiscernables ($d(x, y) = 0 \Leftrightarrow x = y$). The *diameter* $\Delta_d(S)$ of a set of points S in a metric space is the maximum distance between any pair of points: $\Delta_d(S) = \max_{p,q \in S} d(p, q)$.

Definition 2.1 (Metric Functional Dependency (MFD)): Given a relation r defined over a relation schema R, attribute sets $X, Y \subseteq R$, a metric d over Y, and a parameter $\delta \geq 0$, the *metric functional dependency* $X \xrightarrow{\delta} Y$ is said to hold if

$$\max_{T \in \pi_X} \Delta_d(T[Y]) \leq \delta$$

Example 2.1: Consider again the example in Table I. Let $X =$ TITLE, and Y be DURATION. One set $T \in \pi_X$ is the pair of tuples {(`movie-vault.com`, A Beautiful Mind, 144), (`walmart.com`, A Beautiful Mind, 145)}. $T[Y]$ is the set $\{144, 145\}$, which under the Euclidean metric has diameter $\Delta_d(T[Y]) = 1$. Similar computations for the other elements of π_X yield the values 2 and 6 for the corresponding diameters. Thus, the MFD $X \xrightarrow{6} Y$ holds over this relation.

III. VERIFYING METRIC FDS

It is straightforward to verify whether the FD $X \longrightarrow Y$ exists. For each class $T \in \pi_X$, we merely check that $|T[Y]| = 1$. For metric FDs, we need more complex algorithms. In this section, we study the verification problem for MFDs. Specifically, in this section we study the complexity of verifying whether the MFD $X \xrightarrow{\delta} Y$ holds for a given relation r. Note that the more general problem, of finding *some* choice of parameters for which the dependency holds, can be reduced to the verification problem via binary search.

The choice of metric directly impacts the inherent complexity as well as efficiency of the verification procedures, as shown below:

General Metric Spaces: It follows directly from the definition of an MFD (Definition 2.1) that the key subroutine needed to verify the MFD $X \xrightarrow{\delta} Y$ is a procedure that computes the diameter of a set of points in a metric space. Let $n = |r|$ be the size of the relation.

Observation 3.1: In $O(n^2)$ time we can verify whether the MFD $X \xrightarrow{\delta} Y$ holds. Further, in $O(n)$ time we can verify whether there exists $\delta' \leq 2\delta$ such that $X \xrightarrow{\delta'} Y$ holds.

Proof: The first statement follows trivially from a simple brute force procedure. The second statement follows from the following algorithm: Pick any point and find its furthest neighbour. It is well known [11] that this distance is at least half the optimal diameter; doubling it gives the desired approximation. ∎

We refer to the brute force approach as BRUTE and the linear time 2-approximation as 2-APPROX.

Euclidean Metrics: If we have no further information about the metric, then 2-APPROX is the best known approach. In practice however, natural metrics on attributes will have more structure. For example, the DURATION field from Table I induces the Euclidean distance $d(a, b) = |a_{\text{DURATION}} - b_{\text{DURATION}}|$. For this metric, diameter computation is trivial: compute the maximum and minimum values and take their difference. For points in the plane, diameter computation can be performed in $O(n \log n)$ time by computing the convex hull of the points and walking along the boundary in linear time [12] using the rotating calipers method. We refer to this algorithm as CALIPERS.

It is possible to compute the diameter in $O(n \log n)$ time for points in three dimensions [13]; however, this algorithm is highly impractical. For $d \geq 4$, there can be $\Omega(n^2)$ diameter pairs, and so obtaining better exact solutions is difficult, although there are some approaches. A more useful direction is to allow the algorithm to return an *approximate diameter*, which might be within a factor of $1 + \alpha$ of the correct answer. This will immediately yield an approximate verifier for the MFD $X \xrightarrow{\delta} Y$; as we saw in the case of general metric spaces, we will be able to verify whether there exists a $\delta' \leq (1 + \alpha)\delta$ such that $X \xrightarrow{\delta'} Y$.

The best known approach (in terms of dependence on n) for computing the approximate diameter of a set P of n points in R^d is the *core-set*-based approach of Agarwal *et al.* [14]. Their algorithm yields the desired approximation in time $O(n + 1/\alpha^{\frac{3(d-1)}{2}})$. A variant of this algorithm was implemented by Yu *et al.* [15] and was demonstrated to be quite effective in low dimensions $d \leq 8$. We use this algorithm for approximate diameter calculation and refer to it as CORESET. For details of the algorithm, we refer the reader to [15].

String Metrics: For string data, an appropriate metric is the distance derived from the cosine similarity of high-dimensional q-gram vectors, obtained from the q-gram frequency counts of strings. The normalized q-gram vectors can be considered as points lying on a high-dimensional unit sphere. The cosine similarity between any two such points can be evaluated as the dot product of the normalized vectors. Thus, computing the cosine dis-similarity of q-gram vectors is equivalent to computing the Euclidean distance between corresponding high-dimensional (normalized) vectors, and thus the algorithm CORESET from above can be used.

However, this algorithm does not scale well with dimension, and thus it is more effective to use the algorithms BRUTE and 2-APPROX, by treating the high dimensional vectors as points in a general metric space. In Section IV, we present experimental results for this approach.

The Verification Algorithm: Algorithm 1 summarizes the overall verification procedure. The procedure DIAM(P, α) invokes the appropriate algorithm from above, returning a number r within a $(1+\alpha)$ factor of the diameter of P. If $\alpha = 0$, the exact diameter is returned. The algorithm makes a linear number of calls to DIAM(P, α), one for each $T \in \pi_X(r)$.

IV. EXPERIMENTS

Experiments were performed on a 2.7 GHz dual-core Pentium PC with 2 GB of RAM. The performance results presented are based on real time as reported by the Unix `time` command. Each experiment was repeated 10 times and the average time was reported. All algorithms were implemented in C++. We used three data sources for our experiments. Table IV displays a summary of data characteristics as well as the metrics used and FDs tested.

DataSet	Dimension	#Tuples	Avg #Consequents/Antecedent	Source	Metric	FD Tested
MovieDuration	1	16688	33	Web	Absolute Difference (L_1)	Title $\longrightarrow$ Duration
LatLong	2	100000	100	Synthetic	Euclidean Distance (L_2)	Address $\longrightarrow$ (Latitude, Longitude)
Strings	676	100006	200	DBGen	Cosine Similarity	SSN $\longrightarrow$ Name

TABLE IV
Summary Statistics For Our Data Sources

Input: Relation r, attribute sets X, Y, parameter δ, error parameter $\alpha \geq 0$

Output: YES if $X \overset{(1+\alpha)\delta}{\longrightarrow} Y$, else NO

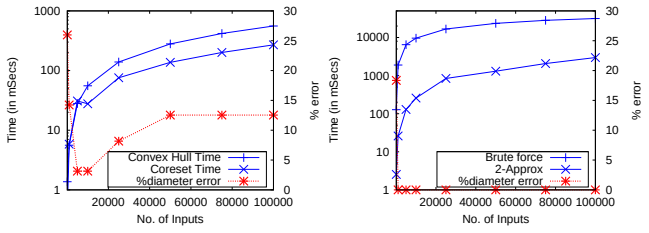
1: **for** each $T \in \pi_X(r)$ **do**
2: Let $\delta_T \leftarrow \text{DIAM}(T[Y], \alpha)$
3: **if** $\delta_T > (1+\alpha)\delta$ **then**
4: Return NO
5: **end if**
6: **end for**
7: Return YES

Algorithm 1. VerifyMFD: Verifying $X \overset{\delta}{\longrightarrow} Y$

We study experimental results of Algorithm VerifyMFD for verifying MFDs.

One dimension: As discussed in Section III, the diameter of a set of points on the line can be computed trivially by reporting the minimum and maximum along the axis.

Two dimensions: We implement Algorithm Calipers using *QHull* [16] for convex hull construction and rotating calipers [17] to compute diameter of the convex hull. Algorithm Coreset is implemented using the method of [15], with code provided by Hai Yu. Figure 1(a) compares the performance and quality of the aforementioned approaches with euclidean similarity metric. The left Y-axis of Figure 1(a), plotted in log-scale, compares the time complexity of Calipers against Coreset. For large number of inputs, Coreset significantly outperforms Calipers. The right Y-axis presents the % error in diameter calculation. As can be seen, the error introduced due to Coreset is around 10%-15% for large input sizes.



(a) (b)

Fig. 1. Performance and quality comparison of: (a) Calipers vs Coreset for LatLong (b) Brute vs 2-Approx for Strings

Higher dimensions: Figure 1(b) compares the performance and quality of Brute and 2-Approx with cosine similarity metric for Strings. The left Y-axis of Figure 1(b), plotted in log-scale, compares the running time of Brute against 2-Approx. As expected, with increase in number of inputs, linear time 2-Approx outperforms quadratic complexity Brute. More importantly, the right Y-axis which presents the % error in diameter calculation shows that the error in diameter calculation is zero for most input sizes.

V. Discussion

Classical dependency theory applies to traditional databases, and much current work in this realm involves extending notions of dependency to be more robust to data failure and errors. In this paper, we have introduced the idea of a metric functional dependency as a way of validating data relationships robustly when dealing with heterogeneous data sources. Our results show that the notion is fundamentally sound and realistic: dependency verification can be performed on large data sets, and for a diverse collection of possible data types.

References

[1] "Geocoding," http://stevemorse.org/jcal/latlon.php.
[2] E. Rahm and H. H. Do, "Data cleaning: Problems and current approaches," *IEEE Data Eng. Bull.*, vol. 23, no. 4, pp. 3–13, 2000.
[3] J. Kivinen and H. Mannila, "Approximate inference of functional dependencies from relations," *Theor. Comput. Sci.*, vol. 149, no. 1, pp. 129–149, 1995.
[4] C. Giannella, "An axiomatic approach to defining approximation measures for functional dependencies," in *ADBIS '02: Proceedings of the 6th East European Conference on Advances in Databases and Information Systems*. London, UK: Springer-Verlag, 2002, pp. 37–50.
[5] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis, "Conditional functional dependencies for data cleaning," in *ICDE*. IEEE, 2007, pp. 746–755.
[6] G. Cong, W. Fan, F. Geerts, X. Jia, and S. Ma, "Improving data quality: consistency and accuracy," in *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*. VLDB Endowment, 2007, pp. 315–326.
[7] W. Fan, "Dependencies revisited for improving data quality," in *PODS*, M. Lenzerini and D. Lembo, Eds. ACM, 2008, pp. 159–170.
[8] M. Arenas, L. E. Bertossi, and J. Chomicki, "Consistent query answers in inconsistent databases," in *PODS*. ACM Press, 1999, pp. 68–79.
[9] J. Chomicki, J. Marcinkowski, and S. Staworko, "Computing consistent query answers using conflict hypergraphs," in *CIKM*, D. Grossman, L. Gravano, C. Zhai, O. Herzog, and D. A. Evans, Eds. ACM, 2004, pp. 417–426.
[10] A. Fuxman and R. J. Miller, "First-order query rewriting for inconsistent databases," *J. Comput. Syst. Sci.*, vol. 73, no. 4, pp. 610–635, 2007.
[11] P. Indyk, "Sublinear time algorithms for metric space problems," in *STOC*, 1999, pp. 428–434.
[12] G. T. Toussaint, "Solving geometric problems with the rotating calipers," in *Proceedings of IEEE MELECON*, 1983.
[13] E. A. Ramos, "Deterministic algorithms for 3-d diameter and some 2-d lower envelopes," in *SoCG*, 2000, pp. 290–299.
[14] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan, "Approximating extent measures of points," *J. ACM*, vol. 51, no. 4, pp. 606–635, 2004.
[15] H. Yu, P. K. Agarwal, R. Poreddy, and K. R. Varadarajan, "Practical methods for shape fitting and kinetic data structures using core sets," in *Symposium on Computational Geometry*, J. Snoeyink and J.-D. Boissonnat, Eds. ACM, 2004, pp. 263–272.
[16] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, "The quickhull algorithm for convex hulls," *ACM Trans. Math. Softw.*, vol. 22, no. 4, pp. 469–483, 1996, http://www.qhull.org/download/.
[17] "Opencv library," http://opencvlibrary.sourceforge.net/.