Maintaining Consistency of Probabilistic Databases: A Linear Programming Approach

You Wu and Wilfred Ng

Department of Computer Science and Engineering
The Hong Kong University of Science and Technology
Hong Kong, China
cs_wyxab@stu.ust.hk, wilfred@cse.ust.hk

Abstract. The problem of maintaining consistency via functional dependencies (FDs) has been studied and analyzed extensively within traditional database settings. There have also been many probabilistic data models proposed in the past decades. However, the problem of maintaining consistency in probabilistic relations via FDs is still unclear. In this paper, we clarify the concept of FDs in probabilistic relations and present an efficient chase algorithm $LPChase(r, \mathcal{F})$ for maintaining consistency of a probabilistic relation r with respect to an FD set $\mathcal{F}$. $LPChase(r, \mathcal{F})$ adopts a novel approach that uses Linear Programming (LP) method to modify the probability of data values in r. There are many benefits of our approach. First, $LPChase(r, \mathcal{F})$ guarantees that the output result is always the minimal change to r. Second, assuming that the expected size of an active domain consisting data values with non-zero probability is fixed, we demonstrate the interesting result that the LP solving time in $LPChase(r, \mathcal{F})$ decreases as the probabilistic data domains grow, and becomes negligible for large domain size. On the other hand, the I/O time and modeling time become stable even when the domain size increases.

1 Introduction

There have been a rapid growth of important applications that operate on probabilistic data, such as sensor monitoring network and mobile object tracking. The data in these applications are inherently uncertain due to signal noises or instrumental errors. Various recent work on probabilistic DBMSs has been developed to support the management of uncertain data [3, 12, 8].

Integrity constraints ensure that changes made to the database do not result in a loss of data consistency. Functional dependencies (FDs) are known to be the most fundamental integrity constraints in relational databases [9]. Thus, repairing data inconsistency with respect to FDs has become an important problem as studied in the context of crisp databases such as [14]. However, there still lacks study of using FDs to maintain consistency of probabilistic databases.

In this paper, we study how to maintain consistency of a probabilistic relation when given a set of FDs. Our main objectives are twofold. First, we clarify the concept of a consistent probabilistic relation r with respect to $\mathcal{F}$. We introduce Manhattan distance measure, rather than Euclidean distance measure, in defining the semantics of an FD and show that Manhattan distance captures better our intuition of probabilistic difference between data. Second, we aim at developping an effective and efficient means to modify a probabilistic relation from inconsistent status to consistent status. By effectiveness we mean that the method should only impose minimal possible changes on r. By efficiency we mean that the method should be efficient when the probabilistic domains and other database parameters scale up.

Informally, the problem is described as follows: *Given a probabilistic relation r and a set of functional dependencies $\mathcal{F}$, we modify the probability of the data values in r such that the modified relation r' is consistent with respect to $\mathcal{F}$. There is no insertion or deletion of tuples in r in the process of modification.*

Table 1: An example of a probabilistic relation r

	A	B	C	*Tuple Frequency*
t_1	<0, 1.0>	<0, 0.3>, <1, 0.7>	<0, 0.1>, <1, 0.9>	1
t_2	<0, 1.0>	<1, 0.5>, <2, 0.5>	<0, 0.6>, <2, 0.4>	2
t_3	<1, 0.7>, <2, 0.3>	<2, 1.0>	<0, 1.0>	3
t_4	<3, 0.8>, <4, 0.2>	<3, 1.0>	<3, 1.0>	4

Table 1 shows a probabilistic relation r having four tuples t_1 to t_4, and three attributes A, B and C, in which each attribute value is associated with a probability distribution on a set of data values $\{0, 1, 2, 3, 4\}$. (We only show data values having non-zero probability.) The last column describes the occurring frequency of a tuple in r. Assume a given FD $A \rightarrow B$. As the first two tuples are identical over A, they should be identical over B as well, but they are not indeed. We may modify the attribute values B of both tuples to {<0, 0.1>, <1, 0.567>, <2, 0.333>}, thus making these two tuples *consistent* with respect to $A \rightarrow B$. This modification is the *minimal* possible one (and the only possible one) if we want to maintain the relative value frequency of each attribute value over attribute B (see Def. 4). It essentially changes the probability distribution on the involved data values in $\{0, 1, 2\}$. Notably, r is general enough to address the features of probabilistic data models with attribute or tuple level uncertainty.

Our main contributions are threefold.

- We define the consistency problem having minimal possible change on probabilistic relations and call the problem MCP. We transform MCP into a standard LP problem. This approach of maintaining consistency in probabilistic relations with respect to $\mathcal{F}$ is novel, effective and efficient.
- Given an inconsistent relation r and an acyclic FD set $\mathcal{F}$, we present a polynomial time algorithm $LPChase(r, F)$ that returns the best possible consistent relation, in the sense that the minimal change of r is guaranteed.
- We verify our complexity analysis on $LPChase(r, \mathcal{F})$ by a set of experiments on various database parameters. We show the interesting result that the running time of $LPChase(r, \mathcal{F})$ decreases as the probabilistic data domain size grows, and becomes nearly negligible for large domain size. On the other hand, the I/O and modeling time is stable even if the domain size increases.

The rest of the paper is organized as follows. Formal definitions of our probabilistic model and other necessary background concepts are given in Sect. 2. In Sect. 3, we clarify the notion of FD satisfaction in r. In Sect. 4, we define the Minimal Consistency Problem (MCP) and explain how to solve the problem by LP transformation. In Sect. 5, we present a polynomial time chase algorithm. Experiment results will be shown in Sect. 6. In Sect. 7, we discuss the related work. Finally, we conclude our work in Sect. 8.

2 Background

In this section, we formally define probabilistic relations and introduce a vectorial representation for data domains. We denote by $\mathbb{R}$ the set of real numbers and by $\mathbb{Z}$ the set of integers throughout the paper.

Definition 1. (Probabilistic Data Domain) *Given a data domain D. Let $E_D = \{(v,p) \mid v \in D$ and $p \in [0,1] \cap \mathbb{R}\}$ be a probability distribution over D. E_D is said to be valid if all the v values of $(v,p) \in E_D$ are pairwise distinct and $\sum_{(v,p)\in E_D} p = 1$. A probabilistic data domain of D, denoted by D_{pr} is the set of all valid E_D.*

The requirement that all the p values for a valid E_D should sum up to 1 is important for data consistency. This also conforms to the possible world semantics for attribute level uncertainty model discussed in [2].

For simplicity in presentation, we assume in this paper all data domains are denoted by D with $|D| = k$ such that there is a linear order on D. However, it is straightforward to generalize the subsequent results by using heavier notation if distinct domains are considered.

Example 1. {<1, 0.3>, <2, 0.7>} is a valid E_D for $D = \{1, 2\}$, since 0.3+0.7=1, while {<1, 0.3>, <2, 0.6>} and {<1, 0.3>, <1, 0.7>} are not, since the former has 0.3+0.6=0.9$\neq$1 and the latter has the same v value equal to 1.

Definition 2. (Probabilistic Relation) *A probabilistic tuple (or simply a tuple) t over a relational schema R with respect to D is a tuple where $\forall A \in R$, $t[A] \in D_{pr}$. Each tuple is attached with a non-negative integer value F_t (tuple frequency). A probabilistic relation (or simply a relation) r over R with respect to D is a finite set of tuples over R. The total frequency T of r is given by $T = \sum_{t\in r} F_t$. The relative frequency of a tuple $t \in r$ is given by F_t/T.*

The concept of tuple frequency in r is useful in many scenarios involving uncertain data. This represents the importance of readings from uncertain sources.

Next, we introduce a vectorial representation of an attribute value in r, which helps to manipulate the tuples.

Definition 3. (Vectorial Representation) *The vectorial representation of $t[A]$, where t is a tuple over a relational schema R and $A \in R$, denoted by $\boldsymbol{V_{t,A}}$, is $(p_1, p_2, \ldots, p_k)$, with p_i being the probability of the ith value $v_i \in D$.*

Given an attribute A, we define how likely a value in domain D appears by using tuple frequencies.

Definition 4. (Relative Value Frequency) *Let r be a relation over R and $A \in R$, $\boldsymbol{V_A} = \sum_{t \in r} \boldsymbol{V_{t,A}} \cdot F_t/T = (f_{1,A}, f_{2,A}, \ldots, f_{k,A})$. For $i \in [1,k] \cap \mathbb{Z}$, $f_{i,A}$ is the relative value frequency of domain value v_i over attribute A.*

Example 2. Table 2 shows the corresponding vectorial representation of the probabilistic relation r shown in Table 1. We assume domain $D = \{0,1,2,3,4\}$. For example, $t_1[B] = (0.3, 0.7, 0, 0, 0)$ means that this probability distribution consists of five value-probability pairs, namely, <0, 0.3>, <1, 0.7>, <2, 0>, <3, 0> and <4, 0>. It can be checked that $t_1[B]$ is a valid probability distribution, since the probabilities for the first two non-zero probability pairs are 0.3 and 0.7, which are added up to 1. We also have $\boldsymbol{V_A} = (0.3, 0.21, 0.09, 0.32, 0.08)$.

Table 2: An example of a vectorial representation of a probabilistic relation r

	A	B	C	$\frac{F_t}{T}$
t_1	(1, 0, 0, 0, 0)	(0.3, 0.7, 0, 0, 0)	(0.1, 0.9, 0, 0, 0)	0.1
t_2	(1, 0, 0, 0, 0)	(0, 0.5, 0.5, 0, 0)	(0.6, 0, 0.4, 0, 0)	0.2
t_3	(0, 0.7, 0.3, 0, 0)	(0, 0, 1, 0, 0)	(1, 0, 0, 0, 0)	0.3
t_4	(0, 0, 0, 0.8, 0.2)	(0, 0, 0, 1, 0)	(0, 0, 0, 1, 0)	0.4

3 FUNCTIONAL DEPENDENCY

In this section, we introduce a distance measure for comparing tuples and relations, and define satisfaction of FDs in a relation.

In conventional relational model, a functional dependency (FD) is in the form of $X \rightarrow Y$. Two tuples t_1 and t_2 are said to satisfy the FD $X \rightarrow Y$ if $t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y]$. We follow the same spirit and propose that if two probabilistic tuples are "probabilistically similar" over X, they should also be "probabilistically similar" over Y. In literature, Euclidean distance is commonly adopted when measuring the distance between uncertain data objects, such as [6]. However, we argue that the Euclidean distance is not a good distance measure used for maintaining consistency in probabilistic relations. We now illustrate this point with the following example.

Example 3. Let the size of D be 200. Let $t[A]$ and $t'[A]$ be two tuples projected over attribute A. Let $\boldsymbol{V_{t,A}} = (p_1, p_2, \ldots, p_{200})$ and $\boldsymbol{V_{t',A}} = (p'_1, p'_2, \ldots, p'_{200})$, where $\forall i \in [1,100] \cap \mathbb{Z}$ and $\forall j \in [101,200] \cap \mathbb{Z}$, $p_i = p'_j = 0, p_j = p'_i = 0.01$. It is easy to verify that the normalized Euclidean distance (ranges between 0 and 1, inclusive) between $t[A]$ and $t'[A]$ is given by

$$Dist_{Euc}(t[A], t'[A]) = \sqrt{0.5 \cdot 0.01^2 \cdot 200} = 0.1.$$

Clearly, the normalized Euclidean distance is too small to convey the fact that the probability distributions of $t_1[A]$ and $t_2[B]$ have no common data value having non-zero probability. The result strongly violates the intuition that the distance between these two disjoint E_D should be far from each other.

We now introduce the concept of normalized Manhattan distance, which intuitively expresses the change in probability between E_D more accurately.

Definition 5. (Manhattan Distance: Single Attribute) *The distance between two tuples t_1 and t_2 over an attribute A with respect to D is given by*

$$Dist(t_1[A], t_2[A]) = \frac{1}{2}\sum_{i=1}^{k} |p_{1,i} - p_{2,i}|,$$

where $k = |D|$ and $\boldsymbol{V_{t_1,A}} = (p_{1,1}, p_{1,2}, \ldots, p_{1,k})$, $\boldsymbol{V_{t_2,A}} = (p_{2,1}, p_{2,2}, \ldots, p_{2,k})$ are the vectorial representations of $t_1[A]$ and $t_2[A]$ respectively.

Proposition 1. *The following statements are true.*
1. $0 \le Dist(t_1[A], t_2[A]) \le 1$.
2. $Dist(t_1[A], t_2[A]) = 1$ *if and only if* $t_1[A]$ *and* $t_1[A]$ *have no common value with non-zero probability.*
3. $Dist(t_1[A], t_2[A]) = 0$ *if and only if* $t_1[A]$ *and* $t_2[A]$ *are exactly the same.*

The following example illustrates the distance measure defined in Def. 5.

Example 4. Consider Example 3 again. $Dist(t[A], t'[A]) = \frac{1}{2} \cdot 0.01 \cdot 200 = 1$. This example shows that normalized Manhattan distance reflects the distance between two disjoint E_D more accurately.

We need to generalize the distance measure for the case of multiple attributes.

Definition 6. (Manhattan Distance: Multiple Attributes) *The distance between two tuples t_1 and t_2 over a set of attributes X is given by*

$$Dist(t_1[X], t_2[X]) = max_{A \in X}\{Dist(t_1[A], t_2[A])\}.$$

The distance we adopted enjoys the property of triangle inequality. As will be shown in our later analysis, the modeling time is dominating the running time in the chase algorithm. Using this property, reference points can be used to identify similar tuples so that the number of comparisons will be greatly reduced, thus saving the modeling time.

Proposition 2. *Manhattan distance satisfies triangle inequality:*

$$Dist(t_1[X], t_2[X]) \le Dist(t_1[X], t_3[X]) + Dist(t_3[X], t_2[X]).$$

We now formally define the notion of an FD.

Definition 7. (Functional Dependency) *A Functional Dependency (FD for short) over R is a statement of the form $X_\alpha \to Y_\beta$, where $X, Y \subseteq R$ and $\alpha, \beta \in [0,1] \cap \mathbb{R}$. Given a relation r over R, if $\forall t_1, t_2 \in r$,*

$$Dist(t_1[X], t_2[X]) \le \alpha \Rightarrow Dist(t_1[Y], t_2[Y]) \le \beta,$$

we say FD $X_\alpha \rightarrow Y_\beta$ is satisfied in r, or equivalently, r is consistent with respect to $X_\alpha \rightarrow Y_\beta$. Given a set of FDs $\mathcal{F}$, we say $\mathcal{F}$ is satisfied in r, or equivalently, r is consistent with respect to $\mathcal{F}$ if and only if r satisfies every FD in $\mathcal{F}$.

Given an FD $X_\alpha \rightarrow Y_\beta$, whether two tuples are similar over X and/or Y is determined by the two parameters α and β. Our model reduces to conventional relational model when $\alpha = \beta = 0$ and all E_D values contain a single value with probability equal to 1. The setting of α and β in an application depends on the specific setting of the application or the discretion of domain experts.

In this paper, we assume all FDs are non-trivial, that is, for a given FD $X_\alpha \rightarrow Y_\beta$, $Y \cap X = \emptyset$. The following result justifies that we only need to consider single attribute on the right-hand side of an FD in subsequent discussion.

Proposition 3. *Decomposition rule holds for FD satisfaction:*

Given r, r satisfies $X_\alpha \rightarrow Y_\beta$ iff r satisfies $X_\alpha \rightarrow A_{i\beta}$ for all $A_i \in Y$.

Def. 7 does not involve tuple frequencies. Thus, we modify the probability of data values in tuples of r in order to fix inconsistency. A tuple $t \in r$ is said to be modified to t' whenever the probability of data values in some attributes are changed. However, as will be shown later in our chase procedure, tuples with higher frequencies (heavier tuples) will be modified less. This brings us the benefit of imposing minimal change for the whole relation.

The following definition will be used to measure the effectiveness of our algorithm in terms of the distance between the input relation r and its modified result r'. Note that F_t is taken into consideration to define the distance but the tuple frequencies remain the same throughout the modification, i.e. $F_{t'_i} = F_{t_i}$.

Definition 8. (Distance Between Modified Relations) *Let r be modified to r'. We assume a fixed linear order imposed on the tuples of r, i.e. $(t_1 > t_2 > \cdots > t_n)$ in r and $(t'_1 > t'_2 > \cdots > t'_n)$ in r', where $t'_i \in r'$ is the modified tuple corresponding to $t_i \in r$. The distance between r and r' is then given by*

$$Dist(r, r') = c \cdot \sum_{i=1}^{n} \sum_{A \in R} Dist(t_i[A], t'_i[A]) \cdot F_{t_i}$$

where constant $c = \frac{1}{|R| \cdot T}$.

4 MINIMAL CONSISTENCY PROBLEM

In this section, we briefly review the underlying idea of Linear Programming (LP) technique and define the Minimal Consistency Problem (MCP) in relations. We show how to solve MCP by transforming the problem into standard LP setting.

4.1 Linear Programming Algorithm

Linear programming (LP) is a technique that optimizes a linear objective function, subject to a given set of linear equality and linear inequality constraints. The standard form (or the canonical form) of a linear programming problem is that, given a variable vector $\overrightarrow{X} = (x_1, x_2, \ldots, x_n)$, a constant vector $\overrightarrow{C} = (c_1, c_2, \ldots, c_n)$ and a linear objective function $G = \overrightarrow{C}^T \cdot \overrightarrow{X}$ which is subject to a set of constraints expressed in a matrix equation $\mathbf{M} \cdot \overrightarrow{X} \leq \overrightarrow{B} (or \geq \overrightarrow{B})$ where $\mathbf{M}$ is an $m \times n$ matrix with constant entries and $\overrightarrow{B}$ is a constant vector $(b_1, b_2, \ldots, b_m)$, we are able to optimize G by using the *simplex algorithm*. We employ an *LP solver lp_solve* detailed in [10] to tackle the consistency problem in this work.

4.2 Minimal Consistency Problem and LP Transformations

In subsequent discussion, we denote r the input relation and r' the output relation where r and r' conform to the same schema. The variables $x_{i,j}$ and $x'_{i,j}$ will carry the same meaning as defined in Defs. 9, 10, 11 and 12.

We now define the problem of generating a consistent relation with respect to a given FD $\mathcal{F}$ where the right-hand side of all FDs is A and the left hand side of no FD has A. Suppose we modify r to r'. There are several requirements for the modification. First, the modification should change nothing over attribute values of X. Second, r' should satisfy $\mathcal{F}$. Third, it does not change the tuple frequencies in r. Fourth, it does not change the relative value frequency of any domain value over attribute A (recall Def. 4). Finally, r' should differ as little as possible from r according to the distance measure in Def. 8. We formalize all the requirements of the modification and call the problem MCP.

Definition 9. (Minimal Consistency Problem MCP). *Let X be a finite set of attributes and A be a single attribute. Let $\mathcal{F} = \{X_{i\alpha_i} \rightarrow A_{\beta_i} | X_i \subseteq X\}$ be a finite set of FDs. Let k be the size of domain of attribute A. Let $\mathbf{V_{t_i,A}} = (x_{i,1}, x_{i,2}, \ldots, x_{i,k})$ and $\mathbf{V_{t'_i,A}} = (x'_{i,1}, x'_{i,2}, \ldots, x'_{i,k})$, $i = 1, 2, \ldots, n$. The Minimal Consistency Problem (MCP) is to find r' that minimizes $Dist(r, r')$ and satisfies the following four conditions (C1 to C4).*

C1. $r[X] = r'[X]$.
C2. *r' is consistent with respect to $\mathcal{F}$.*
C3. $\forall i \in [1, n] \cap \mathbb{Z}, F_{t_i} = F_{t'_i}$.
C4. $\forall i \in [1, k] \cap \mathbb{Z}, f_{i,A} = f'_{i,A}$.

In the above definition, *C1* to *C4* correspond to our first four requirements. The last requirement is realized by minimizing $Dist(r, r')$ as already stated. Let T denote the sum of tuple frequencies of all tuples of r, *i.e.*, $T = \sum_{i=1}^{n} F_{t_i}$. We now transform MCP into the first version of its equivalent LP problem. We start by assuming the special case $\mathcal{F} = \{X_\alpha \rightarrow A_\beta\}$ (i.e. singleton $\mathcal{F}$).

Definition 10. (1LPMCP: First LP Transformed MCP). *We minimize the objective function: $\frac{1}{(|X|+1) \cdot T} \cdot \sum_{i=1}^{n} (\sum_{j=1}^{k} |x'_{i,j} - x_{i,j}|) \cdot F_{t_i}$, which is subject to the following constraints:*

1. $\forall i \in [1, n] \cap \mathbb{Z}$, $\sum_{j=1}^{k} x'_{i,j} = 1$.

2. $\forall i \in [1,n] \cap \mathbb{Z}, j \in [1,k] \cap \mathbb{Z},\ 1 \geq x'_{i,j} \geq 0.$
3. $\forall j \in [1,k] \cap \mathbb{Z},\ \sum_{i=1}^{n} x'_{i,j} \cdot F_{t_i} = \sum_{i=1}^{n} x_{i,j} \cdot F_{t_i}.$
4. $\forall i,j \in [1,n] \cap \mathbb{Z},\ \frac{1}{2}\sum_{l=1}^{k} |x'_{i,l} - x'_{j,l}| \leq \beta$ *if* $Dist(t_i[X], t_j[X]) \leq \alpha.$

Notably, the four conditions in Def. 9 are addressed in the above definition: *C1* and *C4* in Def. 9 are satisfied, since all the involved values in these two conditions remain constant in the LP formulation. *C3* is realized by the third 1LPMCP constraint and finally *C2* is realized by the fourth 1LPMCP constraint. The objective function of 1LPMCP is employed to find minimal $Dist(r, r')$. The first two 1LPMCP constraints in Def. 10 simply ensure the modified relation r' is a valid one.

There is still a problem in 1LPMCP: we need to convert the absolute expressions of the fourth 1LPMCP constraint and the objective function into standard linear form in order that 1LPMCP can be solved by a linear programming algorithm. To achieve the conversion, we need to introduce more technical variables and define another version of LPMCP as follows.

Definition 11. (2LPMCP: Second LP Transformed MCP) *We minimize the objective function:* $\frac{1}{(|X|+1)\cdot T} \cdot \sum_{i=1}^{n}(\sum_{j=1}^{k} d_{i,j}) \cdot F_{t_i}$*, which is subject to the following constraints:*

1. $\forall i \in [1,n] \cap \mathbb{Z},\ \sum_{j=1}^{k} x'_{i,j} = 1.$
2. $\forall i \in [1,n] \cap \mathbb{Z}, j \in [1,k] \cap \mathbb{Z},\ 1 \geq x'_{i,j} \geq 0.$
3. $\forall j \in [1,k] \cap \mathbb{Z},\ \sum_{i=1}^{n} x'_{i,j} \cdot F_{t_i} = \sum_{i=1}^{n} x_{i,j} \cdot F_{t_i}.$
4. $\forall i,j \in [1,n] \cap \mathbb{Z},$
   a. $\forall l \in [1,n] \cap \mathbb{Z},\ D_{i,j,l} \geq x'_{i,l} - x'_{j,l}$ *and* $D_{i,j,l} \geq x'_{j,l} - x'_{i,l}.$
   b. *if* $Dist(t_i[X], t_j[X]) \leq \alpha,\ \frac{1}{2}\sum_{l=1}^{k} D_{i,j,l} \leq \beta.$
5. $\forall i \in [1,n] \cap \mathbb{Z}, j \in [1,k] \cap \mathbb{Z},\ d_{i,j} \geq x'_{i,j} - x_{i,j}$ *and* $d_{i,j} \geq x_{i,j} - x'_{i,j}.$

In Def. 11, $d_{i,j}$ and $D_{i,j,l}$ are the newly introduced variables that replace the absolute expressions in Def. 10. Clearly, the objective function and all the constraints in 2LPMCP are in standard setting of a linear programming problem. Therefore, the minimal value of the objective function can be obtained by using the simplex algorithm.

The following theorem formally shows that the minimal value of the 2LPMCP objective function is equal to the minimal possible distance $Dist(r, r')$ in MCP.

Theorem 1. *Let* m_1 *be the minimal value of the objective function in 2LPMCP. Let* m_2 *be the minimal possible distance* $Dist(r, r')$ *in MCP. Then* $m_1 = m_2$*.*

By Theorem 1, we establish the result that MCP having a single FD ($X_\alpha \rightarrow A_\beta$) can be solved in polynomial time. Next, we consider transforming MCP to the general case of multiple FDs having the same attribute on the right-hand side.

Definition 12. (3LPMCP: Third Transformed MCP). *We minimize the objective function:* $\frac{1}{(|X|+1)\cdot T} \cdot \sum_{i=1}^{n}(\sum_{j=1}^{k} d_{i,j}) \cdot F_{t_i}$*, which is subject to the following constraints:*

1. $\forall i \in [1,n] \cap \mathbb{Z}$, $\sum_{j=1}^{k} x'_{i,j} = 1$.
2. $\forall i \in [1,n] \cap \mathbb{Z}, j \in [1,k] \cap \mathbb{Z}$, $1 \geq x'_{i,j} \geq 0$.
3. $\forall j \in [1,k] \cap \mathbb{Z}$, $\sum_{i=1}^{n} x'_{i,j} \cdot F_{t_i} = \sum_{i=1}^{n} x_{i,j} \cdot F_{t_i}$.
4. $\forall i, j \in [1,n] \cap \mathbb{Z}$
   a. $\forall l \in [1,k] \cap \mathbb{Z}$, $D_{i,j,l} \geq x'_{i,l} - x'_{j,l}$ *and* $D_{i,j,l} \geq x'_{j,l} - x'_{i,l}$.
   b. $\frac{1}{2} \sum_{l=1}^{k} D_{i,j,l} \leq min_{X_{p\alpha_p} \to A_{\beta_p} \subseteq \mathcal{F}}$ $\{[[Dist(t_i[X_p], t_j[X_p]) \leq \alpha_p]] \cdot \beta_p + (1 - [[Dist(t_i[X_p], t_j[X_p]) \leq \alpha_p]]) \cdot +\infty\}$.
5. $\forall i \in [1,n] \cap \mathbb{Z}, j \in [1,k] \cap \mathbb{Z}$, $d_{i,j} \geq x'_{i,j} - x_{i,j}$ *and* $d_{i,j} \geq x_{i,j} - x'_{i,j}$.

The essential difference between 3LPMCP and 2LPMCP is in the right-hand side of the inequality in Constraint *4b*. The square bracket $[[E]]$ is a notation that returns 1 if the boolean expression E is evaluated to be true, otherwise 0. The expressions in the constraint contain only known values, so the right-hand side is still a constant expression. The following corollary immediately follows from Theorem 1.

Corollary 1. *Let m_1 be the minimal value of the objective fucntion in 3LPMCP. Let m_2 be the minimal possible distance $Dist(r, r')$ in MCP. Then $m_1 = m_2$.*

From Corollary 1, we can see that 3LPMCP is an effective means to obtain a consistent relation r' modified from r, since it guarantees minimal possible change from the original (inconsistent) relation r with respect to $\mathcal{F}$.

Complexity Note. The worst case running time of the simplex algorithm is exponential. However, Spielman and Teng [13] used smoothed analysis to show that the algorithm has a polynomial complexity in terms of the input size and magnitude of perturbation, where small random perturbation can be caused by noises or instrumental errors. In our problem setting, the smoothed complexity for solving 3LPMCP is $O(n^8k^4 \ln nk)$. The modeling time for transforming the input of MCP to the input of 3LPMCP is $O(n^2k||\mathcal{F}||)$. Corollary 1 implies that solving 3LPMCP is equivalent to solving MCP. Therefore, the total time for solving MCP is $O(n^8k^4 \ln nk + n^2k||\mathcal{F}||)$.

4.3 Further Improving the evaluation of 3LPMCP

The time for solving 3LPMCP ($O(n^8k^4 \ln nk)$) dominates the modeling time ($O(n^2k||\mathcal{F}||)$) asymptotically. However, the running time for solving 3LPMCP can be further reduced as follows.

We construct an undirected weighted graph G where each vertex v_i corresponds to tuple t_i of input relation r for $i = 1, 2, \ldots, n$. An edge of weight d exists between two vertices v_i and v_j if and only if $d (\leq 1)$ is the maximum distance allowed between t_i and t_j according to $\mathcal{F}$ (cf. Constraint *4b* in Def. 12). It is clear that two tuples in different connected components of G will not violate any FD in $\mathcal{F}$. We partition r into sub-relations where tuples belong to the same sub-relation if and only if their corresponding vertices are in the same connected component of G. Then 3LPMCP of the original relation r can be solved by solving 3LPMCP of all the sub-relations.

In fact, the running time for solving 3LPMCP becomes negligible compared to I/O time and modeling time as domain size increases, because each subrelation becomes smaller and the coefficient matrix of 3LPMCP is very sparse (most entries are zero). This interesting point will be further discussed using the empirical results presented in Sect. 6.

5 CHASE ALGORITHM: LPCHASE

Chase algorithms are a commonly used technique to deal with consistency problems in databases [9, 14]. Our chase also takes a relation r and an FD set $\mathcal{F}$ as inputs and outputs a consistent relation r' with respect to $\mathcal{F}$. In contrast to others, our chase algorithm is developed by solving 3LPMCP which guarantees minimal change to r.

Before presenting our chase algorithm, we define a specific class of FD sets, called *acyclic* FD sets. We will show that acyclic FD sets are a necessary assumption in constructing the algorithm.

Definition 13. (Acyclic FDs). *Let $\mathcal{F}$ be a set of FDs over a relational schema R. Let $G_{\mathcal{F}} = (V, E)$ be a directed graph, where $V = R$ and $E = \{(A, B) | \exists X_\alpha \rightarrow Y_\beta \in \mathcal{F}$, such that $A \in X$ and $B \in Y\}$. $\mathcal{F}$ is said to be acyclic iff $G_{\mathcal{F}}$ is acyclic.*

Notably, the definition of an acyclic FD set is more general than that of a canonical FD set [7]. Obviously, a canonical set should be acyclic. However, the reverse is not true. For example, $\mathcal{F} = \{A \rightarrow B, B \rightarrow C\}$ is acyclic but it is not canonical, since B occurs on both the left-hand side and the right-hand side.

We now present our chase algorithm based on solving MCP in Algorithm 1. This algorithm, denoted by LPChase($r, \mathcal{F}$), takes a relation r and an acyclic FD set $\mathcal{F}$ as inputs. In Algorithm 1, MCPSolve($\mathcal{F}'', A, r[\boldsymbol{X}A]$) is defined to be a function that chases $r[\boldsymbol{X}A]$ over FD set $\mathcal{F}''$ by modifying tuples over attribute A, MCPSolve uses the 3LPMCP model described in Sect. 4.

Algorithm 1 LPChase($r, \mathcal{F}$)

```
 1: $V \leftarrow R$
 2: $E \leftarrow \{(A, B) | \exists (X_\alpha \rightarrow B_\beta) \in \mathcal{F}, such\ that\ A \in X\}$
 3: $(A_1, A_2, \ldots, A_{|R|}) \leftarrow TopoSort(V, E)$
 4: $\mathcal{F}' \leftarrow Decompose(\mathcal{F})$
 5: for $i = 1$ to $|R|$ do
 6:     $\mathcal{F}'' \leftarrow \{(X_\alpha \rightarrow A_{i\beta}) \in \mathcal{F}'\}$
 7:     if $\mathcal{F}'' \neq \emptyset$ then
 8:         $\boldsymbol{X}_i \leftarrow \bigcup \{X | \exists X_\alpha \rightarrow A_{i\beta} \in \mathcal{F}'\}$
 9:         $MCPSolve(\mathcal{F}'', A_i, r[\boldsymbol{X}_i A_i])$
10:         $\mathcal{F}' \leftarrow (\mathcal{F}' - \mathcal{F}'')$
11:     end if
12: end for
13: return $r$
```

The underlying idea of LPChase is that we first generate the acyclic graph (V, E) from the input FD set $\mathcal{F}$ (acyclic FDs) in Lines 1 and 2 and impose an topological order ($A_1 < A_2 < \cdots < A_{|R|}$) on the attributes of the schema R by $TopoSort(V, E)$ in Line 3. Then in Line 4 we decompose $\mathcal{F}$ into $\mathcal{F}'$ s.t. all FDs in $\mathcal{F}'$ contain only one attribute on their right-hand sides. The "for" loop from Lines 5 to 12 is a *chase* that essentially checks the consistency of r with respect to the set of FDs according to the topo order of the right-hand-side attribute A_i. In each round the subrelation $r[\boldsymbol{X}_i A_i]$ and all the FDs having A_i on the right-hand side are passed to MCPSolve($\mathcal{F}'', A_i, r[\boldsymbol{X}_i A_i]$) in Line 9. The remaining FDs ($\mathcal{F}' - \mathcal{F}''$) which may contain A_{i+1} on the right-hand side are generated in Line 10 for preparing the chase of next round.

The following theorem presents important properties of LPChase.

Theorem 2. *The following statements are true.*

(1) $\forall i \in [1, k] \cap \mathbb{Z}$, $f_{i,A}$ (the relative value frequency of domain value v_i) does not change in LPChase($r, \mathcal{F}$).
(2) r is consistent with respect to $\mathcal{F}$ after LPChase($r, \mathcal{F}$).
(3) LPChase($r, \mathcal{F}$) terminates in polynomial time.

Example 5. Given r over $\{A, B, C, D\}$ and an FD set $\mathcal{F} = \{A_{\alpha_1} \to B_{\beta_1}, B_{\alpha_2} \to CD_{\beta_2}, AB_{\alpha_3} \to D_{\beta_3}\}$. We have $\mathcal{F}' = Decompose(\mathcal{F}) = \{A_{\alpha_1} \to B_{\beta_1}, B_{\alpha_2} \to C_{\beta_2}, B_{\alpha_2} \to D_{\beta_2}, AB_{\alpha_3} \to D_{\beta_3}\}$. According to Def. 13, $G_{\mathcal{F}}$ can be topologically sorted as $ABCD$ (another possibility is $ABDC$). Thus, Algorithm 1 chases for the first attribute $A_1 = A$ but gets $\mathcal{F}'' = \emptyset$ in round 1. It chases for $A_2 = B$ and gets $\mathcal{F}'' = \{A_{\alpha_1} \to B_{\beta_1}\}$, $\boldsymbol{X}_2 = A$ and $\mathcal{F}' = \{B \to C, B \to D, AB \to D\}$. It chases for $A_3 = C$ and gets $\mathcal{F}'' = \{B_{\alpha_2} \to C_{\beta_2}\}$, $\boldsymbol{X}_3 = B$ and $\mathcal{F}' = \{B_{\alpha_2} \to D_{\beta_2}, AB_{\alpha_3} \to D_{\beta_3}\}$. Finally, it chases for $A_4 = D$ and gets $\mathcal{F}'' = \{B_{\alpha_2} \to D_{\beta_2}, AB_{\alpha_3} \to D_{\beta_3}\}$, $\boldsymbol{X}_4 = AB$ and $\mathcal{F}' = \emptyset$.

6 EXPERIMENTS

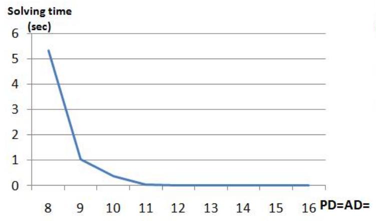
We implemented Algorithm 1 in Microsoft Visual C++. All experiments were conducted on a Windows-based machine with 2.66GHz CPU and 1GB RAM. We use a set of synthetic data generated according to Gaussian distribution. For LP algorithm, we use the *lp_solve* library [10] as discussed in Sect. 4.1.

The main goal in our experiments is to study the efficiency of LPChase. Specifically, we examine the running time against various database parameters related to data domain, tuples and FD sets. The total running time of LPChase consists of three main components: modeling time that generates the 3LPMCP setting as described in Def. 12, I/O time that transfers tuples to memory and MCP solving time that runs the LP solver.

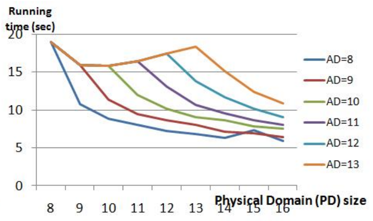
6.1 Domain Size k

In this study, we observed that the number of data values that have non-zero probabilities in the data domain D is important to the running time of LPChase. To understand the impact better, we define the set of all data values that have non-zero probability in r the *active domain* (or simply AD), which is computed by the expected size of AD derived from Gaussian distribution. We call D the *physical domain* (or simply PD). Clearly $AD \subseteq PD = D$.

We fix $n = 20000$ and $||\mathcal{F}|| = 2$ (a single FD $A_{0.05} \rightarrow B_{0.05}$) throughout this experiment. As shown in Fig. 1(a), the time for solving the modeled LP decreases drastically as k increases and become negligible around $k = 10$ (i.e. $PD = AD = 10$). As k is normally much larger than 10, the time for solving MCP is negligible. The total running time then becomes linear to k due to the modeling time in LP transformation (recall the complexity note in Sect. 4.2). The running time of Algorithm 1 for various ADs is shown in Fig. 1(b). It unanimously shows that when k (i.e. PD) is larger than AD, the running time decreases as k increases. This is because when the PD size is increased, with all other parameters unchanged, tuples are less likely to be similar. Hence, each sub-relation (i.e. $r[\boldsymbol{X}A_i]$ used in Line 9 of Algorithm 1) is small and thus the sum of all the MCPSolve running time (each is quadratic to n) decreases. Intuitively, this gain is due to the "divide-and-conquer" strategy as explained in Sect. 4.3.



(a) Solving time for MCP



(b) Total running time

Fig. 1: Running time of LPChase with different domain size k

6.2 Size of FDs $||\boldsymbol{F}||$

We fix $n = 20000$, $k = 10$ and $\alpha = \beta = 0.05$ for each FD in this experiment. Fig. 2 shows the running time of Algorithm 1 with different sizes of FD set. The running time is linear with $||\mathcal{F}||$, since the modeling time is linear to $||\mathcal{F}||$ and the time for solving LP is negligible compared to I/O and modeling time.

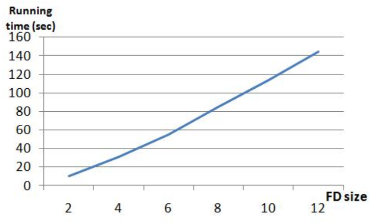


Fig. 2: Running time of LPChase with different $||\mathcal{F}||$

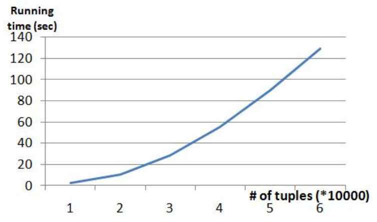
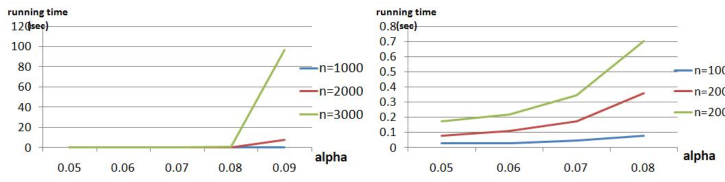


Fig. 3: Running time of LPChase with different n

6.3 Size of Relation n

We fix $k = 10$, $||\mathcal{F}|| = 2$ and $\alpha = \beta = 0.05$ for each FD in this experiment. Fig. 3 shows the running time of LPChase with different number of tuples. The running time is non-linear, since the time for solving LP is negligible at $k = 10$ compared to the I/O time (in $O(n)$) and the modeling time (in $O(n^2)$) and thus the modeling time, which is non-linear, is dominating.

6.4 Sensitivity of FD α



(a) Total running time (b) Time for solving MCP

Fig. 4: Running time of LPChase with different α

We fix $k = 10$ and $||\mathcal{F}|| = 2$ in this experiment. Fig. 4 shows the running time against different α values when $n = 1000, 2000$ and 3000 respectively. As shown in Fig. 4(a), when α increases from 0.08 to 0.09, the running time increases significantly. The running time is very sensitive to α at the critical point (0.08-0.09 in Fig. 4). Fig. 4(b) shows the total running time of LPChase ($\alpha = 0.05 - 0.08$). It is clear that the sizes of 3LPMCP sub-problems increase monotonically with the increase of α and so does the running time. However, in this experiment setting, k is only 10 (i.e. PD = AD = 10). When k becomes larger, non-zero probability values are distributed over more domain values and the critical point of α will increase rapidly because tuples are less likely to be similar. Admittedly, this is an issue deserved further study. We may perform a binary search on α, model the 3LPMCP sub-problems for a fixed α without solving them, and examine the sizes of the sub-problems until an acceptable α value is found.

7 RELATED WORK

The problem of maintaining the consistency of a conventional relational database is well-known [14]. When dealing with uncertain information (data are missing, unknown, or imprecisely known), probability theory, fuzzy set and possibility theory-based treatments have been applied to extend classic relational databases. For example, stochastic dependency [5] is the generalization of the concept of functional dependency in probabilistic databases, and fuzzy functional dependencies [1] have been proposed to handle the integrity constraints problem in the context of fuzzy databases. Demetrovics proposed the error-correcting functional dependency [4] in a deterministic database containing erroneous data. However, the mentioned work is for checking data consistency rather than developing an effective and efficient means to maintain consistency for uncertain data. Levene [9] introduces the notion of imprecise relations and employs FDs to maintain imprecise relations. An imprecise data model is to cater for relational data obtained from different equally likely and noise-free sources, and therefore may be imprecise. Lu and Ng [11] use *vague sets* that assume data sources are not equally likely to address similar issues.

8 CONCLUSIONS

We have studied the problem of maintaining consistency of probabilistic relations with respect to acyclic FD sets, which are stronger than the canonical form of FDs [7]. We developed an LP-based chase algorithm, called LPChase, which can be employed to maintain the consistency by transforming MCP into an LP setting. LPChase has a polynomial running time and it is efficient in practice if the domain size is not extremely small (say $\mid D \mid= 10$) and the α values are chosen before a critical point. The output of LPChase is also effective in the sense that it is the minimally modified input relation as proved in Corollary 1. As shown in Sect. 6, the time for solving the modeled LP problems becomes negligible compared to the time for constructing them and the I/O time, when the physical domain is reasonably larger than the active domain of r. Then the running time of LPChase is bounded by $O(n^2k||\mathcal{F}||)$.

References

1. P. Brown and P.J. Haas, BHUNT: automatic discovery of fuzzy algebraic constraints in relatoinal data, *VLDB*, 668–679 (2003)
2. G. Cormode, F. Li, and K. Yi, Semantics of ranking queries for probabilistic data and expected ranks, *ICDE*, 305–316 (2009)
3. N. Dalvi and D. Suciu, Efficient query evaluation on probabilistic databases, *VLDB Journal*, 16(4), 523–544 (2007)
4. J. Demetrovics *et al.*, Functional dependencies distorted by errors, *DAM*, 156(6), 862–869 (2008)
5. D. Dey and S. Sarkar, Generalized normal forms for probabilistic relational data, *TKDE*, 14(3), 485–497 (2002)
6. C. Faloutsos and K. I. Lin, FastMap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets, *SIGMOD*, 163–174 (1995)
7. S. Greco and C. Molinaro, Approximate Probabilistic Query Answering over Inconsistent Databases, *ER*, 311–325 (2008)
8. J. Huang *et al.*, MayBMS: a probabilistic database management system, *SIGMOD*, 1071–1074 (2009)
9. M. Levene, Maintaining consistency of imprecise relations, *Comput. J.* 39(2), 114–123 (1996)
10. lp_solve, http://lpsolve.sourceforge.net/5.5/
11. A. Lu and W. Ng, Maintaining consistency of vague databases using data dependencies, *DKE*, 68(7), 622–651 (2009)
12. S. Singh *et al.*, Orion 2.0: native support for uncertain data, *SIGMOD*, 1239–1242 (2008)
13. D.A. Spielman and S.H. Teng, Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time, *JACM*, 51(3), 385–463 (2004)
14. J. Wijsen, Database repairing using updates, *TODS*, 30(3), 722–768 (2005)