Querying Uncertain Data with Aggregate Constraints

Mohan Yang [1,2]*, Haixun Wang [1], Haiquan Chen [3], Wei-Shinn Ku [3]
[1]Microsoft Research Asia, Beijing, China
[2]Shanghai Jiao Tong University, Shanghai, China
[3]Auburn University, Auburn, AL, USA
mh.yang.sjtu@gmail.com,haixunw@microsoft.com,{chenhai,weishinn}@auburn.edu

ABSTRACT

Data uncertainty arises in many situations. A common approach to query processing uncertain data is to sample many "possible worlds" from the uncertain data and to run queries against the possible worlds. However, sampling is not a trivial task, as a randomly sampled possible world may not satisfy known constraints imposed on the data. In this paper, we focus on an important category of constraints, the aggregate constraints. An aggregate constraint is placed on a set of records instead of on a single record, and a real-life system usually has a large number of aggregate constraints. It is a challenging task to find qualified possible worlds in this scenario, since tuple by tuple sampling is extremely inefficient because it rarely leads to a qualified possible world. In this paper, we introduce two approaches for querying uncertain data with aggregate constraints: constraint aware sampling and MCMC sampling. Our experiments show that the new approaches lead to high quality query results with reasonable cost.

Categories and Subject Descriptors

H.2.4 [**DATABASE MANAGEMENT**]: Systems

General Terms

Algorithms, Performance

Keywords

Uncertain Data, Aggregate Constraint, MCMC

1. INTRODUCTION

Probabilistic data appear in numerous applications, including information extraction [19, 20, 32], graph anlytics [15, 22], track and trace [7, 13], data cleaning [6, 7], etc. In this paper, we study the problem of query processing against uncertain data where the data is subject to global constraints, or aggregate constraints.

Assume a dataset has an attribute with uncertain values. For each tuple, we use a probability mass function (pmf), which means it has value x_i with probability p_i, to describe its uncertainty. As a toy example, imagine a company running multiple advertisement campaigns to increase its sales. Table 1 shows estimated sales increase resulting from web, TV, and newspaper advertisements. The values in the *Sales Increase* column are uncertain. The pmf $\{2/0.3, 4/0.7\}$ means that the sales increase is 2 million with probability 0.3, and 4 million with probability 0.7.

ID	**Ad Campaign**	**...**	**Sales Increase (million $)**
1	Web	...	{2/0.3, 4/0.7}
2	TV	...	{3/0.2, 5/0.8}
3	Newspaper	...	{1/1.0}

Table 1: A dataset D with uncertain values.

Possible World W	$\Pr[W]$
{(Web, 2), (TV, 3), (Newspaper, 1)}	$0.3 * 0.2 * 1 = 0.06$
{(Web, 2), (TV, 5), (Newspaper, 1)}	$0.3 * 0.8 * 1 = 0.24$
{(Web, 4), (TV, 3), (Newspaper, 1)}	$0.7 * 0.2 * 1 = 0.14$
{(Web, 4), (TV, 5), (Newspaper, 1)}	$0.7 * 0.8 * 1 = 0.56$

Table 2: Possible worlds.

Clearly, in this toy example, as in many real life applications, the joint distribution of the entire data is not available. We can instantiate a possible world by drawing a value from the pmf associated with each tuple. Table 2 illustrates the possible worlds of the uncertain sales data, where $\Pr[W]$ is the probability of the corresponding possible world W.

With possible worlds, we can answer queries against the uncertain data. For example, to answer the query, "How likely are TV ads more effective than Web ads?", we find possible worlds from Table 2 where the sales increase from TV ads is larger than that from Web ads, and we add up their probabilities, $0.06 + 0.24 + 0.56 = 0.86$. Consequently, the answer is 86%. Since the entire set of possible worlds is often huge (which we will quantify later), a common practice is to sample the possible worlds and answer the queries from the sampled data.

1.1 Constraints

The pmf associated with each tuple reflects that our belief, or prior knowledge, about the data is uncertain. When deriving the possible worlds in Table 2, we assume that tuples in the dataset are not correlated.

This allows us to draw values from a tuple's pmf in a way independent of other tuples, and, hence, we are able to instantiate possible worlds at a low cost. Additional information may change

*Work done while the author was at Microsoft Research Asia.

our belief and the pmf. In particular, the independence assumption may no longer hold, making sampling much more difficult.

Specifically, although uncertainty is associated with each individual record, there exist global statistics or constraints about the entire dataset in many systems [6, 27]. Here are two examples:

- A company is running a series of advertisement campaigns to increase sales. It is known that, after all the campaigns, sales increase by a total amount between \$8 and \$9 million, but the amount contributed by each campaign is uncertain.

- Assume RFID-based sensors are used at highway entrances and exits for track and trace. It is known that the total number of trucks that have passed a certain toll gate is between 180 and 200, but traces of individual trucks (which entrances and exits they use) are uncertain.

These global statistics are called *aggregate constraints*. They occur frequently in real life applications, and they introduce correlations among records. As a result, the cost of instantiating possible worlds from uncertain data may increase dramatically.

1.2 Challenges and Observations

When the number of possible worlds is huge and the constraints are numerous and hard to satisfy, looking for qualified possible worlds may become as hard as finding a needle in a haystack. To illustrate this, assume each tuple has k possible values. Thus, the total number of possible worlds is k^n, where n is the number of tuples in the uncertain dataset. For example, if $k = 3$ and $n = 1M$, then there are $3^{1,000,000} > 10^{477,121}$ possible worlds. We further assume that the constraints actually disqualify 99.9999% possible worlds, out of the total $10^{477,121}$ possible worlds.

In fact, the problem is known to be #P-hard [23]. Next, we illustrate two observations that are important for designing an efficient sampling strategy.

1.2.1 Constraint Aware Sampling is More Efficient

When no aggregate constraint is present, one can sample each tuple based on its pmf, independent of other tuples. However it is not difficult to see why aggregate constraints make independent sampling extremely inefficient. Given a constraint, if we instantiate tuples one by one, there is no way to know if the resulting dataset satisfies the constraint until we instantiate the last few tuples. Furthermore, when a set of tuples are influenced by more than one constraint, it is even more difficult for a randomly sampled possible world to be a valid possible world. Thus, in order to reduce the rejection rate, one may want to change the pmf according to which we are drawing the values for each tuple. In other words, we want to modify the pmf's in such a way that they become constraint aware, i.e., sampling from the new pmf's will result in a possible world that is more likely to satisfy the aggregate constraints.

1.2.2 Valid Possible Worlds Tend to Cluster Together

Figure 1 illustrates the entire sampling space. Here, each point in the space represents a possible world. In this setting, a random point only has 0.000001 (one out of a million) chance of being a valid possible world.

One possible approach is to adopt a method used in constraint satisfaction problems; that is, given a random point in the space, to find a path that efficiently leads to a nearby valid point. As we will show later, traditional constraint satisfaction approaches have high complexity and cannot handle problems in our scale. Furthermore, even if the constraints disqualify 99.9999% of the possible worlds, there are still more than $10^{477,121} \cdot 10^{-6} = 10^{477,115}$ valid ones. To create even a very small sample set, we need to repeat the above process a large number of times, which is extremely costly.

One important observation which we make is that valid samples tend to cluster together, which means, given a possible world X that satisfies all the constraints, a possible world Y that is very "similar" to X is very likely to satisfy the constraints as well. In fact, we can "construct" such a Y from a given X. First, pick a tuple from the dataset, if the tuple is not involved in any constraint, then randomly change its value; otherwise, change its value such that the constraints are still satisfied[1]. The resulting possible world Y is also valid.

For this reason, Figure 1 depicts valid possible worlds as clustered "islands". Figure 2 zooms in on an island and shows the sampling strategy. Based on observation, when A is the current qualified possible world, one should sample around A for the next qualified possible world. This may give B or C, and we reject B immediately because it is off the island (unqualified), and we accept C based on a certain strategy such that the resulting set of possible worlds constitutes an unbiased sample.

1.3 Our Approaches

In this paper, we propose two approaches, constraint aware sampling and Markov Chain Monte Carlo sampling, in order to find valid possible worlds.

1.3.1 Constraint Aware Sampling method

Drawing values directly from the pmf associated with each tuple is unlikely to produce valid possible worlds, since the drawing ignores the constraints. In this approach, based on the constraints, we modify each pmf such that drawing values from the modified pmf's are likelier to produce valid possible worlds. We essentially convert the original constraint satisfaction problem into a quadratic programming (QP) problem. We then solve the quadratic programming problem to get the biased probability distribution.

1.3.2 Markov Chain Monte Carlo (MCMC) method

We employ a Monte Carlo method to find possible worlds of high probabilities and this method has two phases. In the first phase, we utilize a greedy constraint satisfaction approach to find some initial valid possible worlds. The approach uses a voting algorithm to determine the direction of value change (downward or upward) for each variable. In the second phase, we perform a Monte Carlo walk on a Markov chain that starts with the first valid assignment and ends when enough samples are acquired. This method performs significantly better than both the constraint aware sampling method and naive sampling method.

[1]This is often possible because an aggregate constraint usually has a valid range, for example, when sales increase is between \$8 to \$9 million. A change in a single tuple's value usually will not make it go out of the range.

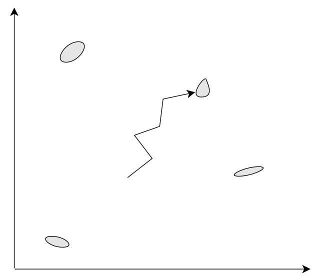

Figure 1: Global search: find islands.

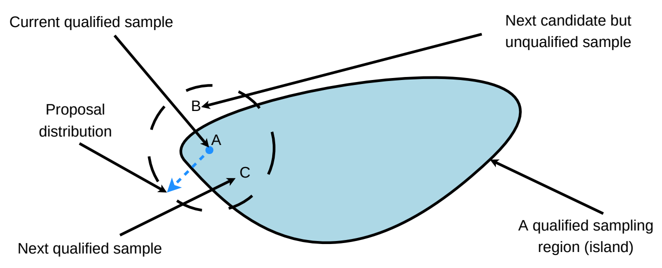
Current qualified sample
Next candidate but unqualified sample
Proposal distribution
B
A
C
A qualified sampling region (island)
Next qualified sample

Figure 2: Local search: sample on each island.

1.4 Paper Organization

The rest of the paper is organized as follows: Section 2 introduces background knowledge of this work and compares our work with related work. Section 3 presents the problem definition and an overview of our approach. In Section 4, we present our constraint aware sampling approach for querying against uncertain data. In Section 5, we propose a more efficient MCMC based approach. Section 6 reports experimental results. We conclude this research in Section 7.

2. RELATED WORK

2.1 Sampling Based Data Management

Many systems have been developed to manage data with uncertainty. The usual approach to address uncertainty is to augment the classical relational model with tuple-level probability values [1, 2, 4, 8, 9]. In the *independent tuples model* [3], an uncertain relation is a collection of tuples. Each tuple is comprised of one or more mutually-exclusive alternatives and associated with a probabilistic mass function. In other words, for a single tuple, if $\sum$ is the sum of the weights or probabilities of all the alternatives of the tuple, then $\sum = 1$. Besides, a possible world of an uncertain relation is a regular relation generated by selecting one and only one alternative from each tuple. Therefore, the total number of possible worlds of an uncertain relation is the product of the number of alternatives in every tuple. Correspondingly, each possible world has an associated probability, which is the product of the probabilities of all the selected alternatives.

In the above approaches, both the uncertainty representation as well as the the data model that determine the uncertainty are fixed. In other words, evidence derived from new incoming data will not be able to alter existing beliefs. Then, in order to incorporate this new evidence, the existing model, that is the entire uncertainty information "hard wired" with the data content, must be changed [14]. In many cases, the logical and physical database designs are inferred from the existing probabilistic uncertainty model, which means the entire system must be overhauled.

A more general approach based on sampling is proposed for managing incomplete and uncertain data [7, 14, 31]. The idea is simple and intuitive: we construct random samples while observing the prior statistical knowledge and constraints about the data. Thus, each sample is one possible realization (possible world) in the space of uncertainty, and the entire set of samples reveals the distribution of the uncertain data which we want to model. Queries and inferences are then conducted against this distribution. MCDB [14], for example, allows a user to define arbitrary variable generation functions that embody the database uncertainty. MCDB then employs these functions to pseudorandomly generate realized values for the uncertain attributes and evaluates queries over the realized values.

One obstacle of applying the sampling approach for managing uncertain and incomplete data is its efficiency. Indeed, with naive sampling (sampling randomly and rejecting a sample if it does not conform with existing statical knowledge or constraints), when the distribution it samples from is complicated, it can be extremely inefficient since a majority of the samples will be rejected. However, advanced sampling techniques, such as sequential importance sampling (SIS), Markov Chain Monte Carlo (MCMC) sampling, which endeavor to sample from regions of high probabilities, can be used to dramatically improve the performance of sampling. For instance, Chen et al. [7] employ the Metropolis-Hastings sampling to approximate the joint distribution efficiently for subsequent query processing.

2.2 Incorporating Aggregate Constraints

How would additional evidences affect the distribution of possible worlds in an uncertain database and change the results of queries over that database? Koch et al. [17] investigate this problem and name it *conditioning*. In *conditioning*, possible worlds that do not satisfy the given constraints are deleted and queries are evaluated over the remaining possible worlds only. Generally, with *conditioning*, we incorporate *evidences* in the form of constraints into the system, and refine a prior probabilistic database to a posterior probabilistic database. For example, in data cleansing techniques, we usually first have an uncertain dataset and then need to clean it (i.e., reduce uncertainty) by applying newly discovered knowledge.

Unfortunately, past research efforts on *conditioning* only focus on imposing functional dependencies in an uncertain database. However, for a set of data, a lot of information or evidence about the data exist in the form of aggregate constraints. For example, statistics about the data may be published by a third party and such statistics are nothing more than aggregates.

The benefit of leveraging aggregate constraints in reducing data uncertainty has been shown in recent research [5, 28]. In [5], the *deduplication* accuracy is improved significantly by incorporating aggregate constraints while [28] exploits domain constraints to enhance the quality of matching entities. However, their approaches only focus on the problem of *entity matching* and, due to efficiency issues, do not scale well in our scenario where the number of aggregate constraints may be very large and the number of involved variables (entities) can be extremely huge.

Besides, Ré et al. [23] investigates how to evaluate positive conjunctive queries with predicate aggregates (i.e., HAVING queries) on probabilistic databases. However, for each HAVING query in their research problem, there is at most one aggregate constraint (HAVING predicate) imposed on the dataset.

In this paper, we focus on a more general problem: how to sample efficiently from large-scale uncertain databases with tremendous variables, given an enormous number of aggregate constraints (up to 1 million variables and $10K$ aggregate constraints are involved in our experiments).

2.3 The Constraint Satisfaction Problems

In this subsection we discuss constraint satisfaction problems (CSP), which can be regarded as our problem in a much smaller scale. Tsang details on this problem and lays out a solution framework using search techniques like constraint propagation and local search [30]. Constraint propagation techniques infer that certain values cannot be part of certain variable domains any more because they violate the given constraints. The constraint propagation algorithm works as follows: when a variable X changes its value, the algorithm recomputes the possible domain expression of each variable Y dependent on X. Afterwards, the algorithm generates a new value for Y. If this newly assigned value is different from the previous one, the algorithm will recompute the values for further downstream variables. This process continues recursively until there are no more changes of values in the network. The constraint propagation techniques can fully solve the constraint satisfaction problem. However, they are rarely used due to efficiency issues.

Local search methods work by iteratively changing the values of a small number of variables in order to increase the number of satisfied constraints. Particularly, for the binary CSPs (also called the Boolean satisfiability problems or SAT), wherein the constraint is represented in the conjunctive normal form (CNF), Selman et al. [24, 25, 26] propose local search strategies GSAT and WalkSat to find satisfying assignments. The general algorithm starts by assigning a random value to each variable. If the assignment satisfies all clauses of the CNF, the algorithm terminates. Otherwise, it selects a certain boolean variable and flips it (changes its value). Specifically, GSAT flips the boolean variable that minimizes the number of unsatisfied clauses while WalkSAT makes flips by first randomly picking a clause that is not satisfied by the current assignment and then picking a variable within that clause to flip. The algorithm may restart with a new random assignment if no solution has been found for too long as a way of getting out from local minima of numbers of unsatisfied clauses [24]. In addition, Spears [29] proposes a simulated annealing algorithm (SASAT) to solve SAT problems. However, all the aforementioned local search methods work only for the Boolean satisfiability problems (SAT) and, consequently, cannot be applied to our research problem, where each variable (tuple) in our uncertain database can have multiple possible values (alternatives).

Kitchen and Kuehlmann [16] describe a MCMC based algorithm for sampling solutions to mixed boolean and finite domain CSPs. Their algorithm starts from a random assignment and then performs a Metropolis-Hastings move to a neighboring assignment. The Metropolis-Hastings algorithm generates a random walk using a proposal density and a method for rejecting proposed moves. If the assignment is valid, the algorithm terminates; otherwise it loops to conduct either a local search move or a Metropolis-Hastings move until the assignment is valid. However, their approach assumes that the CPSs have a large solution space and are relatively easy to satisfy. On the contrary, the research in [12] provides a dynamic programming-based solution for determining the number of solutions of finite domain CSPs by utilizing XOR constraints. Unfortunately, it suffers from extreme inefficiency in our research problem because dynamic programming works poorly when coping with a huge number of variables and constraints.

3. PROBLEM DEFINITION

In this section, we formally describe the problem which we plan to solve. Furthermore, we introduce methods to partition the set of constraints to reduce the complexity of the problem.

3.1 Problem Statement

Given a dataset D of uncertain values, we represent an aggregate constraint c in the following form:

$$c = \{S, t, a, b\}$$

where $S \subseteq D$ is the subset of data that the aggregate constraint is imposed on, t is the type of aggregate, and a and b are the minimal and maximum bound of the constraint. The most frequently used aggregates are the five standard database aggregates: *count, sum, max, min*, and *avg*[2]. For instance, let S be the set of promotional campaigns. Constraint $c = \{S, sum, 10, 11\}$, or equivalently:

$$10 \le \sum_{\mathbf{d} \in S} \mathbf{d}.contribution \le 11$$

This means the total campaign benefit to sales is between 10 and 11 million. In the rest of this paper, we focus on the *sum* constraint. Solutions to the other four types of constraints *count, max, min, avg* are similar.

Thus, the problem can be stated as follows. Given a dataset D of uncertain values and a set of constraints $\mathcal{C} = \{c_1, c_2, \cdots, c_k\}$, we want to answer database queries against D under constraints $\mathcal{C}$.

3.2 Problem Decomposition

Let $S_{\mathbf{d}}$ denote tuple $\mathbf{d}$'s **correlated set**, which includes tuple $\mathbf{d}$ and any tuple that shares one or more constraints with $\mathbf{d}$:

$$S_{\mathbf{d}} = \{\mathbf{x} | \exists c, \mathbf{x} \in c.S, \mathbf{d} \in c.S\} \cup \{\mathbf{d}\}$$

Clearly, if tuple $\mathbf{d}$ is not covered by any constraint, then $S_{\mathbf{d}} = \{\mathbf{d}\}$. On the other hand, if $\mathbf{d}$ is covered by a constraint that is imposed on the entire dataset, then $S_{\mathbf{d}} = \{d_1, \cdots, d_n\}$. In other words, $\{\mathbf{d}\} \subseteq S_{\mathbf{d}} \subseteq \{d_1, \cdots, d_n\}$.

A constraint is applied to a set of tuples in the database. Given a constraint, we denote the set of the tuples it applies to as its **coverage**, or its **domain**. We want to partition the constraints into disjoint subsets such that constraints with overlapping coverage are in the same subset. Then, the problem of finding valid possible worlds in the entire search space is decomposed into a set of independent problems of finding possible worlds in much smaller search spaces.

As an example, assume the database contains 6 tuples $\{X_1, \cdots, X_6\}$, and there are two constraints in total.

$$\begin{cases} 6 \le X_1 + X_2 + X_3 \le 10 \\ 6 \le X_4 + X_5 + X_6 \le 10 \end{cases}$$

Since their domains do not overlap, we can easily divide the problem into two sub-problems:

$$\begin{aligned} \text{Data} &: \{X_1, X_2, X_3\} \\ \text{Constraint} &: \{6 \le X_1 + X_2 + X_3 \le 10\} \end{aligned}$$

and

$$\begin{aligned} \text{Data} &: \{X_4, X_5, X_6\} \\ \text{Constraint} &: \{6 \le X_4 + X_5 + X_6 \le 10\} \end{aligned}$$

The original problem can be solved by solving the two subproblems and merging their results.

This divide-and-conquer approach corresponds to finding connected components in a graph. Define a graph G, where vertexes are the tuples in the database. If tuple x and y are involved in the same constraints, we create an edge between x and y. Then, we decompose G into connected components G_1, G_2, $\cdots$, G_c. Finally, we solve the optimization problem in each component and then merge their results.

4. CONSTRAINT AWARE SAMPLING

In this section, we introduce our constraint aware sampling approach for query processing against uncertain data.

[2]At first look, *min* and *max* do not seem to be aggregate constraints as they can be simply applied for each tuple. However, if we have a constraint

$$c = \{S, min, 10, 20\}$$

which means the minimum value is in the range of [10,20]. We cannot simply revise the pmf of all involved tuples by removing possible values smaller than 10, because we need to make sure that at least one tuple has a value in the range of [10, 20].

4.1 Overview

For each tuple, instead of drawing uncertain values from its original pmf, we create a trial distribution based on the pmf and the aggregate constraints so that the final possible worlds created by drawing tuples from the trial distribution are more likely to satisfy the constraints.

The approach we use falls in the category of *importance sampling* in statistics. The idea suggests that when sampling, one should focus on the region(s) of "importance" which in our case, means regions that are more likely to satisfy the constraints, so as to save computational resource. However, the samples obtained from the trial distribution may very likely be biased. To correct the bias, we give each sample an *importance weight*, which is equal to the ratio between the probability of the sample in the original distribution and the probability of the sample in the trial distribution.

4.2 Satisfiability of a Constraint

Consider a constraint $l \leq S \leq u$, where S is a sum aggregate, $S = X_1 + \cdots + X_k$, and we sample $X_1, \cdots, X_k$ directly from their original distributions. We show that the difference between the *center* of the constraint ($\frac{l+u}{2}$) and the expected value of S affect the constraint's satisfiability.

For example, consider a constraint $5 \leq S \leq 7$. We can calculate the expected value of S. Assume $E(S) = \overline{X}_1 + \cdots + \overline{X}_k = 8$. The upper bound (ub) of the constraint is smaller than the expected value ($7 < 8$). Clearly, if one assumes that S follows a Gaussian distribution, then more than half of the samples will have $S > 7$, making these samples invalid.

In order to improve sampling efficiency, we can manipulate each pmf so that the constraint is easier to satisfy. In our case, where the center of the constraint is smaller than the expected value of the aggregate, we will oversample the lower value end and under-sample the higher value end of the pmf of each X_i. Clearly, the expected value of S for $X_1, \cdots, X_k$ sampled this way will be smaller than its original expectation 8. However, the resulting distribution may not be the same as the original distribution. We will then correct the bias introduced by the new distribution by giving each sample a weight.

When there are multiple constraints, optimizing for one may make others more difficult to satisfy. In the rest of section, we first consider the case of single constraint, then extend it to the scenario with multiple constraints.

4.3 The Single Constraint Case

Let $l \leq X_1 + \cdots + X_k \leq u$ be the constraint, and let $\{x_{i,1}/p_{i,1}, \cdots, x_{i,m_i}/p_{i,m_i}\}$ be the pmf of X_i. We first introduce a bias in sampling: instead of sampling $x_{i,j}$ with probability $p_{i,j}$, we sample it with probability $w_{i,j} \cdot p_{i,j}$, where $w_{i,j}$ is the bias.

First, in order for the new probability distribution to be a valid distribution, the biases must satisfy

$$\sum_{j=1}^{m_i} w_{i,j} \cdot p_{i,j} = 1 \tag{1}$$

and

$$w_{i,j} > 0 \tag{2}$$

With the biases, the expected value of X_i will be shifted from $\overline{X_i}$ to $w_i\overline{X_i}$, that is:

$$\sum_{j=1}^{m_i} (w_{i,j} \cdot p_{i,j}) x_{i,j} = w_i \overline{X}_i \tag{3}$$

The goal of introducing the bias is to increase the satisfiability of the constraint. A constraint is easier to satisfy if the expected value of the sum is as close to the center of the constraint interval as possible, that is, we want to minimize $|w_1\overline{X}_1 + \cdots + w_k\overline{X}_k - \frac{l+u}{2}|$, or equivalently

$$\left[\frac{2(w_1\overline{X}_1 + \cdots + w_k\overline{X}_k)}{l+u} - 1\right]^2 \tag{4}$$

In order to formulate it as an optimization problem, we must characterize $w_{i,j}$. As a regularizer, we adopt a simple bias – the linear bias – for $w_{i,j}$, i.e., we prescribe that $w_{i,j}$ is a linear function of $x_{i,j}$:

$$w_{i,j} = a_i + b_i x_{i,j} \quad a_i, b_i \text{ are parameters} \tag{5}$$

Combining (1)(3)(5) we have

$$\begin{cases} a_i = \frac{\sum_{j=1}^{m_i} p_{i,j} x_{i,j}^2 - w_i \overline{X}_i^2}{\sum_{j=1}^{m_i} p_{i,j} x_{i,j}^2 - \overline{X}_i^2} \\ b_i = \frac{(w_i - 1)\overline{X}_i}{\sum_{j=1}^{m_i} p_{i,j} x_{i,j}^2 - \overline{X}_i^2} \end{cases} \tag{6}$$

Furthermore, combining (2) and (6), we obtain the following bound for w_i (see Subsection 4.6 for details):

$$\frac{\max\limits_j x_{i,j}\overline{X}_i - \sum\limits_{j=1}^{m_i} p_{i,j} x_{i,j}^2}{\overline{X}_i(\max\limits_j x_{i,j} - \overline{X}_i)} < w_i < \frac{\min\limits_j x_{i,j}\overline{X}_i - \sum\limits_{j=1}^{m_i} p_{i,j} x_{i,j}^2}{\overline{X}_i(\min\limits_j x_{i,j} - \overline{X}_i)}$$

or

$$l(w_i) < w_i < u(w_i) \tag{7}$$

Combing (4) (7) and the original constraint, we convert the biased sampling problem to an optimization problem:

$$\begin{aligned} \text{minimize} \quad & \left[\frac{2(w_1\overline{X}_1 + \cdots + w_k\overline{X}_k)}{l+u} - 1\right]^2 \\ \text{subject to} \quad & l \leq w_1\overline{X}_1 + \cdots + w_k\overline{X}_k \leq u \ \text{ and} \\ & l(w_i) < w_i < u(w_i) \ \ i = 1, \cdots, k \end{aligned} \tag{8}$$

This is a quadratic programming (QP) problem (see Subsection 4.5). After solving w_i's, we derive a_i's and b_i's according to (6). Then we determine each individual w_{ij} using (5). Finally, we draw samples from the biased pmf $\{x_{i,1}/w_{i,1}p_{i,1} \cdots, x_{i,m_i}/w_{i,m_i}p_{i,m_i}\}$ to obtain possible worlds that are more likely to satisfy the constraint.

4.4 Bias Correction

The possible worlds are obtained by drawing values for each of its tuples using a distribution different from the original distribution. This introduces a bias, which, according to the theory of importance sampling, can be corrected by assigning each possible world an importance weight.

Specifically, let $W = \{X_1, \cdots, X_k\}$ be a possible world. Let $\pi(W)$ be the probability of W based on the original distribution, and let $g(W)$ be the probability of W based on the trial distribution. Then, the importance weight of W is

$$w(W) = \pi(W)/g(W) \tag{9}$$

Since $\pi(W)$ and $g(W)$ can be calculated as the product of the probabilities of X_i's, the importance weight of W in (9) comes to $w(W) = \prod_{i=1}^{k} 1/w_{i,i'}$, assuming $X_i = i'$.

Finally, after obtaining a set of possible worlds, $\{W_1, \cdots, W_n\}$, we can use them to answer any query. As an example, let $q(W)$ be the (numerical) result of asking query q in possible world W, the final, unbiased result is then $[w(W_1)q(W_1) + \cdots + w(W_n)q(W_n)] / [w(W_1) + \cdots + w(W_n)]$.

4.5 The Multiple Constraints Case

A tuple may be involved in two constraints that wish to impose conflicting sampling biases on the tuple. Thus, multiple constraints make biased sampling more difficult.

Let $l_i \leq f_i(\cdot) \leq u_i$ be the i-th constraint. Let $\mathcal{W}$ denote the set of all w_i's, and let $l(w_i)$, $u(w_i)$ represent the lower and upper bound of w_i in (7). Let m_i be the biased mean of $f_i(\cdot)$, that is, m_i is the mean of $f_i(\cdot)$ derived on the dataset through biased sampling. The final optimization problem comes to:

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{k} \left(\frac{2m_i}{l_i + u_i} - 1 \right)^2 \\ \text{subject to} \quad & l_i \leq m_i \leq u_i \; i = 1, \cdots, k \text{ and} \\ & l(w_i) < w_i < u(w_i) \;\; \forall w_i \in \mathcal{W} \qquad (10) \end{aligned}$$

Just as (8), (10) is a quadratic programming (QP) problem with respect to w_i's. QP minimizes

$$f(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T Q \mathbf{w} + \mathbf{c}^T \mathbf{w}$$

with respect to $\mathbf{w} \in \mathcal{R}^n$ subject to linear inequality and linear equality constraints. It is easy to see that the matrix Q in our problem is positive definite, so then we can use the *ellipsoid method* to solve the problem in polynomial time [18].

4.6 Proof of the lower and upper bounds of w_i

Here we give a proof for (7). According to the **Cauchy-Schwartz inequality**, we have

$$\sum_{j=1}^{m_i} p_{i,j} x_{i,j}^2 \times \sum_{j=1}^{m_i} p_{i,j} \geq (\sum_{j=1}^{m_i} p_{i,j} x_{i,j})^2$$

i.e.

$$\sum_{j=1}^{m_i} p_{i,j} x_{i,j}^2 \geq \overline{X}_i^2$$

So if $w_i > 1$, then $b > 0$; if $w_i < 1$, then $b < 0$ (see the expression of b). To guarantee that $w_{i,j} > 0$, we need the following

$$\begin{cases} a_i + b_i \min_j x_{i,j} > 0 & \text{if } w_i > 1 \\ a_i + b_i \max_j x_{i,j} > 0 & \text{if } w_i < 1 \end{cases}$$

For the first inequality, denote $\min_j x_{i,j}$ as m, then we have

$$\sum_{j=1}^{m_i} p_{i,j} x_{i,j}^2 - w_i \overline{X}_i^2 + m(w_i - 1)\overline{X}_i > 0$$

i.e.

$$w_i \overline{X}_i (m - \overline{X}_i) > m\overline{X}_i - \sum_{j=1}^{m_i} p_{i,j} x_{i,j}^2$$

As $m = \min_j x_{i,j}$, so $m < \overline{X}_i$ and we have the following upper bound (ub) for w_i

$$w_i < \frac{\min_j x_{i,j} \overline{X}_i - \sum_{j=1}^{m_i} p_{i,j} x_{i,j}^2}{\overline{X}_i (\min_j x_{i,j} - \overline{X}_i)}$$

Similar to the second inequality, we have the following lower bound (lb) for w_i

$$w_i > \frac{\max_j x_{i,j} \overline{X}_i - \sum_{j=1}^{m_i} p_{i,j} x_{i,j}^2}{\overline{X}_i (\max_j x_{i,j} - \overline{X}_i)}$$

5. MCMC SAMPLING

The constraint aware sampling method is theoretically elegant. However, when the size of the dataset and/or the number of constraints becomes very large, it turns into time consuming, as we will show in our experiments (Section 6). The most important reason is that this method needs one to solve a QP problem, but QP problems become intractable when the size of the dataset is very large. Another reason is that each time the method searches for a valid possible world from scratch. In other words, the method finds an independent sequence of possible worlds.

An important observation made in Section 1 is that valid possible worlds tend to cluster together in the search space. This prompts us to find "correlated" possible worlds, which means we do not search for valid possible worlds individually, but rather build the next valid sample based on the current one. This leads to the Markov Chain Monte Carlo (MCMC) approach for sampling.

The approach consists of two steps. In the first step, we find seed points that correspond to valid possible worlds in the search space (Figure 1). In the second step, we perform Monte Carlo walks using the Metropolis-Hastings sampler, which starts from the seed points in order to sample more valid possible worlds (Figure 2).

5.1 Find Seed Points in Search Space

Our goal is to find an initial set of possible worlds that satisfy all the constraints. These possible worlds will be used as seed points for Monte Carlo walks in the next step.

5.1.1 The Constraint Satisfaction Problem

We can formulate the problem as a constraint satisfaction problem. Local search techniques such as GSAT and WalkSat proposed by Selman et al. [24, 25, 26] work well for Boolean satisfiability problems. Let n_v, n_c, and n_p denote the number of variables (i.e., tuples in the dataset), number of constraints, and average number of variables per constraint, respectively. In GSAT and WalkSat, iteratively we choose a Boolean variable and change (flip) its value. The problem is in choosing which variable, and the criterion is to choose the one whose new value minimizes the number of unsatisfied constraints. The time complexity of the process is $\mathcal{O}(n_v \cdot n_c \cdot n_p)$. In our case, we have $n_v = 10^6$, $n_c = 10^4$, and $n_p = 10^4$, making one flip needing at least 10^{14} calculations. Furthermore, Papadimitriou [21] shows that we need $\mathcal{O}(n_v^2)$ flips (for 2-SAT using pure random walk on an arbitrary satisfiable formula) in order to reach a satisfying assignment. Thus, we need $10^{14} \times (10^6)^2 = 10^{26}$ calculations to find a valid possible world.

5.1.2 Parallel Hopping

As we have shown above, the classic solution to the Boolean constraint satisfaction problem does not apply in our case, not only because our variables are not Boolean, but mainly due to the fact that our dataset is too big (i.e., both the number of variables and constraints are large).

We propose a parallel hopping approach for this problem. Instead of flipping the value of one Boolean variable at a time, in our approach, we employ a greedy voting technique to select multiple variables and alter their values at the same time. Furthermore, since our variables are not Boolean, we also need to decide the direction of value change (upward or downward) for each variable.

The Initial States. If there are multiple aggregate constraints with different coverage, multiple valid regions may exist in the search space and each region is a convex polyhedron formed by hyperplanes. Our goal is to reach each region. To achieve this goal, we randomly choose a set of points in the search space, and apply parallel hopping (described below) from each of them, hoping they lead us to all of the valid regions. If, however, there is only one valid region (there is only one constraint, or all constraints have the same coverage), then we can start with the most probable state. That is, we start with the state where each X_i has its most probable value. For example, we let $X_i = 20$, if its pmf is $\{10/0.2, 20/0.6, 30/0.2\}$. The resulting possible world has maximal probability.

Voting. For every aggregate constraint c, we compute the aggregate in the current state of the database. If its value is smaller than the lower bound of c, we *vote* to increase the value of all variables in c. Specifically, we increase the votes of every variable in c by an amount proportional to the difference between c's lower bound and the current value of the aggregate. Similarly, we decrease the votes of every variable in c if the aggregate value is larger than the upper bound of c. A high level description of the voting method can be found in Algorithm 1.

Algorithm 1 VOTING

```
Input: current point $val$ and constraint set $\mathcal{C}$
Output: vote value $vote$ and constraint state $consState$
1: set $vote$ and $consState$ array to all 0
2: for constraint $c \in \mathcal{C}$ do
3:     $sum$ = evaluation of $c$ at $val$
4:     if $sum \le c.a$ then
5:         value of $c$ needs adjust upward($consState[c] = 1$)
6:         for $v \in c.S$ do $vote[v] \leftarrow vote[v] + (c.a - sum)$
7:     else if $sum \ge c.b$ then
8:         value of $c$ needs adjust downward($consState[c] = -1$)
9:         for $v \in c.S$ do $vote[v] \leftarrow vote[v] - (sum - c.b)$
10:    end if
11: end for
12: return $vote$ and $consState$
```

Intuitively, if there is only one constraint, then all involved variables receive consistent votes of either increasing or decreasing. When there are multiple constraints, it is possible that changing a variable in one direction may make some constraints easier to satisfy while it may make other constraints harder to satisfy. The voting mechanism is introduced to solve this problem; variables receiving many positive (negative) votes will go up (down) in value, while variables receiving equal amounts of positive and negative votes will keep their values relatively unchanged.

For each variable, we adjust its value using a probabilistic method. We outlined the method in Algorithm 2.

Algorithm 2 VALUE_ADJUSTMENT (MCMC)

```
Input: direction $d$, variable $v$ and its pmf, current value of $v$ is $u$, penalty parameter $p$
Output: new value of variable $v$
1: if $d = \uparrow$ then
2:     normalize the probability of value with penalty $1 - p$ on the probability of
       values smaller than $u$ and penalty $1 + p$ on the probability of values bigger
       than $u$
3: else if $d = \downarrow$ then
4:     normalize the probability of value with penalty $1 - p$ on the probability of
       values bigger than $u$ and penalty $1 + p$ on the probability of values smaller
       than $u$
5: end if
6: return sampling from the normalized probability distribution
```

In the probabilistic approach, we draw values from v's original distribution, but with a bias. If we have voted to increase v, values that are larger than its current value are up-sampled and values that are smaller than its current values are down-sampled. For example, assume v's pmf is {1/0.2, 2/0.2, 3/0.2, 4/0.2, 5/0.2}, and the current value of v is 3. We create a biased distribution with parameters p, where the down-sampling rate for values less than the current value is $1 - p$, and the up-sampling rate for values more than the current value is $1 + p$. The result is $\{1/0.2(1-p), 2/0.2(1-p), 3/0.2, 4/0.2(1+p), 5/0.2(1+p)\}$, which after normalization (setting $p = 0.5$), comes to {1/0.1, 2/0.1, 3/0.2, 4/0.3, 5/0.3}. Then we draw values from this pmf for v. The value of p does not really matter in this problem, as we have a simulated annealing (SA) like procedure in Algorithm 3. p is set to 0.5 in the experiments.

Algorithm 3 HOPPING

```
Input: initial point $val$, constraint set $\mathcal{C}$, greedy loop number $p_1$, penalty parameter $p_2$
Output: a valid point $val$
1: for $i = 1$ to MAX-TRIES do
2:     set $T \leftarrow$ initial temperature and $loopTimes \leftarrow 0$
3:     repeat
4:         $(vote, consState) \leftarrow Voting(val, \mathcal{C})$
5:         if $loopTimes \le p_1$ then
6:             for each variable $v$ do
7:                 $val[v] \leftarrow$ Value_Adjustment($\uparrow, v, val[v], p_2$) if $vote[v] > 0$
8:                 $val[v] \leftarrow$ Value_Adjustment($\downarrow, v, val[v], p_2$) if $vote[v] < 0$
9:             end for
10:        else
11:            for constraint $c \in \mathcal{C}$ do
12:                if value of $c$ needs adjust upward then
13:                    $val[v] \leftarrow$ Value_Adjustment($\uparrow, v, val[v], p_2$) for $v \in c.S$
                       with $vote[v] = \max_{u \in c.S} vote[u]$
14:                else if value of $c$ needs adjust downward then
15:                    $val[v] \leftarrow$ Value_Adjustment($\downarrow, v, val[v], p_2$) for $v \in c.S$
                       with $vote[v] = \min_{u \in c.S} vote[u]$
16:                end if
17:            end for
18:            if $val$ is a valid point then return $val$
19:            $\Delta \leftarrow$ increment of satisfied constraints
20:            if $\Delta < 0$ then
21:                generate a random number $r \in [0, 1]$
22:                if $r \ge \exp^{-\Delta/T}$ then restore $val$ with its old value at the begin-
                   ning of the loop
23:            end if
24:            $T = T \times Cooling\_factor$
25:        end if
26:        $loopTimes \leftarrow loopTimes + 1$
27:    until $T$ = terminating temperature
28:    $val \leftarrow$ a randomly generated point
29: end for
30: return "No satisfying point found"
```

The Hopping Algorithm. The algorithm is a loop which runs until constraints $\mathcal{C}$ are satisfied. Internally, it has two phases. The first phase is more greedy, where we take big steps in hopping. Specifically, in the first phase, all variables receive positive (negative) votes change their values upward (downward) simultaneously. In this phase, we try to reach a local optimum by hopping almost all the variables together. In the second phase, we perform local hopping. For each constraint that needs adjustment, we adjust the value of the variable that receives the largest absolute number of votes. If the new point is a valid one, we return this point immediately; otherwise the new point is accepted or rejected based on a simulated annealing (SA) like procedure. The acceptance probability is calculated based on the increment of satisfied constraints. The new point is then accepted if a random generated number is smaller than the acceptance probability, otherwise it is rejected. The SA part helps from becoming stuck at local optima. Thus the algorithm can find a valid point in a fixed amount of time. We describe the hopping algorithm in Algorithm 3.

5.2 Explore Valid Regions Using Markov Chain Monte Carlo

5.2.1 *Markov Chain Monte Carlo*

A Markov chain is a stochastic process which consists of possible states of random variables. It can be denoted as a sequence of states $X_1, X_2, X_3, ..., X_n$, which satisfy

$$p(X_{n+1} = x | X_n = x_n, X_{n-1} = x_{n-1}, ..., X_1 = x_1) = p(X_{n+1} = x | X_n = x_n)$$

where $p(x|y)$ is the transition probability from state y to state x. Markov Chain Monte Carlo (MCMC) is a technique to generate samples from the state space by simulating a Markov chain. The formed Markov chain is constructed in such a way that the chain spends more time in the regions with higher importance, i.e., the stationary distribution of the Markov chain is the same as the target distribution. That is, the Markov chain can converge to the target distribution (the posterior) as its equilibrium distribution. From the perspective of Monte Carlo sampling, as the number of samples are sufficiently large, all the samples can become the fair samples from the posterior. Consequently, we are able to approximate the sophisticated target posterior based on deliberately constructing a Markov chain of all the Monte Carlo samples.

5.2.2 The Metropolis-Hastings Sampler

The Metropolis-Hastings (MH) sampler is one of the most used MCMC-based samplers, which generates a sequence of random walks using a proposal density, and it decides on whether to reject the proposed moves using the rejection sampling.

The MH algorithm simulates a Markov chain in which each state X_{t+1} only depends on the immediately previous state X_t. The proposal density $Q(X'|X_t)$ is used to generate a new proposed sample X' depending on the current state X_t. X' is accepted as the next state X_{t+1} with the probability of an acceptance rate α, which can be formalized as (11). $P(X)$ is the probability of state X, i.e., the probability of the possible world X.

$$\alpha = \min\left\{1, \frac{P(X')Q(X_t|X')}{P(X_t)Q(X'|X_t)}\right\} \tag{11}$$

5.2.3 The MCMC walk in valid regions by the Metropolis-Hastings Sampler

We make an important observation in Section 1 that valid possible worlds tend to cluster together in the search space (for a valid possible world, if you slightly change the value of one of its records, it will probably remain valid). Indeed, each valid region is bounded by many hyperplanes forming a convex polyhedron. Thus, if we start from a point in this region, we can easily reach the next valid point by transition from the current point.

To explore a valid region, we perform a Monte Carlo walk by the Metropolis-Hastings sampler (Algorithm 4). The algorithm loops until enough valid points are sampled. During each loop, the values of several variables are changed. The number of the variables that can be changed at a time (in each step of a MCMC walk) is defined as the *step length*. Parameter $stepLength$ is used to specify the *step length* of a MCMC walk, i.e., at most $stepLength$ variables can change their values simultaneously in each step. For each variable, it can only transfer to its neighboring values based on the corresponding pmf. For example, the pmf of u is

$$\{4/0.05, 5/0.1, 6/0.2, 7/0.3, 8/0.35\} \tag{12}$$

and its current value is 6. Then the transition vector is

$$\{4/0, 5/0.1, 6/0.2, 7/0.3, 8/0\}$$

because only 5 and 7 are the neighboring values of the current value

Algorithm 4 MCMC

```
Input: a valid point $val$, constraint set $\mathcal{C}$, total sample number $maxSampling$,
step length $stepLength$
Output: valid point list
1: add $val$ to valid point list
2: $prob \leftarrow$ prior probability of valid point $val$
3: while size of valid point list $< maxSampling$ do
4:     randomly transfer at most $stepLength$ variables to neighboring values based
       on the probabilities
5:     if $val$ satisfies $\mathcal{C}$ then
6:         with probability $\min\{1, \frac{\text{prior probability of } val}{prob}\}$ add $val$ to valid point
           list, set $prob \leftarrow$ prior probability of $val$ and jump to the beginning of the
           loop
7:     end if
8:     restore $val$ with its old value at the beginning of the loop
9: end while
10: return valid point list
```

6. Then the normalized transition vector can be described as

$$\{4/0, 5/0.17, 6/0.33, 7/0.50, 8/0\}$$

which means that u will take the value of 5 with a probability of 17%, the value of 7 with a probability of 50%, and the value of 6 (the original value) with a probability of 33%. After the transition, the algorithm will test if the proposed state (proposal sample) satisfies constraint set $\mathcal{C}$. If it satisfies, the probability of accepting this new sample is given by (11). If the proposal density is chosen as symmetric ($Q(x|y) = Q(y|x)$), (11) can be simplified as

$$\alpha = \min\left\{1, \frac{P(X')}{P(X_t)}\right\}$$

Thus with probability α, the new sample (i.e., a possible world or a point in valid region) is accepted and added into the valid point list; otherwise, the new point is rejected and no change is made in the current valid point list.

Suppose there are only two variables, u and v, in the constraint set $\mathcal{C}$. The pmf of u and v can be formulated as (12). In addition, the current values of u and v are 5 and 6, respectively. Assume proposed values of u and v are 4 and 5, and the resulting proposed sample ($u = 4$ and $v = 5$) is valid. The value of α can be calculated as $\min\{1, \frac{0.05\times0.1}{0.1\times0.2}\} = 0.25$. In other words, the probability of accepting the proposed sample ($u = 4$ and $v = 5$) is 25%.

5.3 Comparison with Constraint Aware Sampling

The MCMC sampling method tries to locate a valid starting point (sample) by *parallel hopping* and then performs a Monte Carlo walk in order to explore the associated valid region. The constraint aware sampling approach tries to adjust the prior distribution and then do a biased sampling from the modified distribution. The biases (weights) are derived by solving a quadratic programming (QP) problem. When the number of variables is small (for example $< 1,000$), QP is tractable and the biased sampling approach works well in experiments (Section 6). When the number of variables is huge (for example around 1 million), QP becomes intractable. Therefore, the biased sampling approach is ineffective with huge datasets. The MCMC sampling method scales better as the number of variables increases. For cases with more than 1 million variables, the MCMC sampling method still works.

6. EXPERIMENTS

In order to investigate the scalability of our proposed methods, we employ seven uncertain datasets with different sizes in our experiments. Each dataset is associated with 4 parameters as defined

N	The total number of variables in a dataset.
N_c	The total number of constraints in a dataset.
$N_{var/cons}$	The maximum number of variables in one constraint. For pseudo-random number generator provided by the C programming language, the mean of the random number should be the middle of the interval, i.e., the average number of variables per constraint is $0.5N_{var/cons}$.
N_{biased}	The number of biased constraints in a dataset. A biased constraint is a special type of constraint in which the upper bound is smaller than the expectation or the lower bound is larger than the expectation. If $\overline{X}_1 + \overline{X}_2 + \overline{X}_3 = 6$, constraint $3 \le X_1 + X_2 + X_3 \le 4$ and $7 \le X_1 + X_2 + X_3 \le 8$ are all biased constraints.

Table 3: Meanings of parameters.

in Table 3. We randomly generate the pmf for each variable, where each variable can take at most 10 values and the difference between the neighboring values is at most 5. We also generate the constraint set $\mathcal{C}$ based on experimental parameters and the pmf of all the variables. The parameters of each dataset are shown in Table 4. All the experiments are conducted on a Microsoft Windows XP system, with a dual core 2.90 GHz AMD X2 245 CPU and 4GB RAM.

Dataset	N	N_c	$N_{var/cons}$	N_{biased}
data 1	1K	1K	0.1K	0
data 2	1K	0.1K	0.1K	10
data 3	1K	0.1K	0.1K	20
data 4	1K	0.1K	0.1K	40
data 5	1K	0.1K	0.1K	80
data 6	1M	10K	1K	20
data 7	1M	10K	10K	20

Table 4: Parameters of our datasets.

6.1 Performance Comparison between Naive, Constraint Aware and MCMC Sampling Methods

We investigate the performance of each method on answering the queries such as "What is the expectation of

$$\sum_{\{\mathbf{d}|\mathbf{d}\in c.S, c\in\mathcal{C}\}} \mathbf{d} \qquad (13)$$

under $\mathcal{C}$?". If we have 10 variables in $\mathcal{C}$, namely $X_1, \cdots, X_{10}$, then the query is "What is the expectation of $\sum_{i=1}^{10} X_i$?". For each method, we acquire 1000 valid possible worlds and then calculate the expectation of (13) under these possible worlds. The naive sampling method randomly generates a possible world and then tests if the possible world is a valid one. The constraint aware sampling method and the MCMC sampling method are described in previous sections.

The constraint aware sampling method will adjust the original pmf only if biased constraints exist. For data 1, which has no biased constraints, the constraints aware sampling method works just as naive sampling. The same sampling result is utilized in the following figures and tables. For the data 2 to 7, the constraint aware sampling method samples from the biased distribution obtained from solving a QP problem and calculates the final result with bias correction using importance weights. Importance weights, in our research problem, are extremely small because they are the products of a series of small numbers. We used the Apfloat package[3] to handle the arithmetic operation between these small numbers. The precision was set to the number of variables in the corresponding dataset. Due to the bias correction procedure, the constraint aware sampling method requires additional time to calculate the final result after all the valid possible worlds are acquired. In addition, solving an exact QP problem by using the constraint aware sampling method may be slow in practice because the exact QP problem is to minimize $f(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T Q\mathbf{w} + \mathbf{c}^T\mathbf{w}$ subject to $A\mathbf{w} \le \mathbf{b}$ and $\mathbf{lb} \le \mathbf{w} \le \mathbf{ub}$. Therefore, we adopt an approximate solution to the QP problem here. Similar to the design in [10, 11], our approximation approach consists of two phases. Specifically, the first phase involves the calculation of a feasible point (if one exists) while the second phase involves the generation of an iterative sequence of feasible points that converge to the solution. Here we solve a linear programming (LP) problem which minimizes -$\sum_i w_i$ subject to $A\mathbf{w} \le \mathbf{b}$ and $\mathbf{lb} \le \mathbf{w} \le \mathbf{ub}$. The solution of the LP problem is a feasible point of the QP problem. In our experiments, the QP problem is coded in C++, imported into MATLAB R2009b, and solved through the MATLAB command **quadprog**. By passing the solution of the LP problem as the starting point, we employ the **quadprog** algorithm in the second phase. Besides, varying the maximum iteration number of the **quadprog** may lead to different sampling results. Therefore, we also investigate the impact of different iteration numbers in Section 6.2. For the rest of the experiments, the maximum iteration number is set to ∞. For the MCMC sampling method, parameter $stepLength$ is set to 10 to achieve a better trade-off between efficiency and accuracy (see details in Section 6.3).

6.1.1 Sampling Efficiency

We evaluate the performance of each method on generating 1K valid samples (samples satisfying the given constraint set). As shown in Table 5, to obtain 1K valid samples, the MCMC sampling method is several orders of magnitude faster than the other two methods with a much less number of total samples used. Taking data 2 as an example, the MCMC sampling method only spent 2.438 seconds while the naive sampling method and the constraint aware sampling method consumed 139.750 seconds and 208.381 seconds, respectively. The advantage of the MCMC sampling method over the other two methods on efficiency comes from the fact that the MCMC sampling maintains the correlation of valid samples, which significantly reduces the time needed to generate the next valid sample satisfying the given constraint set.

On the other hand, as Table 5 illustrates, the constraint aware sampling method outperforms the naive sampling method in terms of the number of total samples used in data 1, 2, 3 and 4. For data 5, which contains 80 biased constrains and 20 unbiased constraints, the time needed for the constraint aware sampling method is more than 24 hours. Furthermore, because the constraint aware sampling method needs to solve a QP problem, it becomes infeasible when the number of variables is increased to 1M as in data 6 and data 7. The "-" symbol means the corresponding method is unable to return an answer for the given query after a reasonable amount of time. In addition, when the number of variables becomes extremely large (such as 1M in data 6 and data 7), the MCMC sampling method turns out to be the only feasible approach.

6.1.2 Sampling Accuracy

Table 6 presents the sampling accuracy of each method with 1K valid samples. As shown in Table 6, the accuracy is measured by

[3]Apfloat: `http://www.apfloat.org`

the percentage error, which can be calculated using $\frac{|r-s|}{s} * 100\%$ where r is a sampling result and s is the standard answer. In our problem, the standard answer is obtained by sampling 10K valid samples using naive sampling method. Consequently, the sampling accuracy measurement is only practical for data 1 and data 2 since the naive sampling method only works for these two datasets.

The naive sampling method samples directly from the original distribution while the constraint aware sampling method samples from a biased distribution which needs to be proportional to the original distribution to have a small variance. On the other hand, the MCMC sampling method samples with a Markov Chain. In general, we would expect that the relationship between the accuracy conferred by these three methods to be "naive" > "CS" > "MCMC". The experimental results in Table 6 reflect the above relationship exactly.

However, as far as the scalability issue is concerned, it is an opposite case. According to the results shown in Table 5, the relationship between the scalability of the above three methods was "MCMC" > "CS" > "naive". The method with the lowest precision ("MCMC") scaled best while the method with the highest precision ("naive") scaled the worst. Therefore, compared to the two other methods, the constraint aware sampling method is more suitable for medium-scale problem. However, for large-scale problems, the MCMC sampling method became the only choice even though it has a relatively higher error.

6.1.3 Hitting the Region with a Higher Importance

Next we focus on the performance of each method with respect to the ability to discover a region with a higher importance. For our research problem, a desired sampling scheme should be able to discover regions with higher importance for estimating the query result in the entire possible world space, i.e., the returned valid sample set should have a higher average probability. Figures 3 and 4 plot the distributions of the valid samples discovered by each method in data 1 and data 2, respectively. Here the X-axis corresponds to the sample value and the Y-axis corresponds to the natural logarithm of the probability of a sample.

As shown in Figures 3 and 4, compared to the naive sampling method and the constraint aware sampling method, MCMC sampling method successfully detects the regions with higher importance (average probability) and draws samples from those regions. The reason is that the Metropolis-Hastings sampler utilized in our MCMC sampling method always tends to accept samples with higher probabilities and is directed to discover regions with higher importance. On the other hand, the naive sampling method outperforms the constraint aware sampling in terms of detecting regions with higher importance for data 2.

Dataset	Method	Expectation	Time(s)	Total Sample
data 1	naive	13852.825	37.219	325756
	CS	13852.825	37.219	325756
	MCMC	13860.883	2.031	3628
data 2	naive	13777.852	139.750	1371225
	CS	13167.850	208.381	4482
	MCMC	14649.546	2.438	5110
data 3	naive	-	-	-
	CS	13250.000	204.964	22343
	MCMC	14418.426	4.031	4881
data 4	naive	-	-	-
	CS	13536.931	228.444	275834
	MCMC	14738.640	3.656	8196
data 5	naive	-	-	-
	CS	-	> 24h	-
	MCMC	14469.884	4.938	13208
data 6	naive	-	-	-
	CS	-	-	-
	MCMC	456898.292	184.588	3886
data 7	naive	-	-	-
	CS	-	-	-
	MCMC	482685.494	1932.094	12035

Table 5: Performance comparison (with 1K valid samples acquired). The method "CS" is short for the constraint aware sampling method. The "Total Sample" column represents the number of samples the algorithm tested in order to acquire 1K valid samples. The constraint aware sampling method worked just as naive sampling in data 1 as data 1 has no biased constraints. The same sampling result is used for the constraints aware sampling method and naive sampling method in data 1.

	Dataset	naive	CS	MCMC
Error	data 1	0.041%	0.041%	0.099%
	data 2	0.017%	4.411%	6.345%

Table 6: Sampling accuracy. The method "CS" is short for the constraint aware sampling method. The same result is used for the naive sampling method and the constraints aware sampling method in data 1.

6.2 Impact of the Iteration Number in the Constraint Aware Sampling Method

In this experiment, we investigate the impact of the iteration number on the performance of the constraint aware sampling method. The QP problem in data 2 converges to the exact solution after 1130 iterations. We vary the iteration number from 0 to 1100 and conduct biased sampling with the weight derived from the approximate solution. The sampling results are shown in Table 7, where the column "QP Time", "Sample Time" and "Calc Time" represent the time used in solving the QP problem, biased sampling, and calculating the final result using the samples, respectively. In addition, the column "Total Time" denotes the sum of the above three columns. As shown in Table 7, both the time needed for solving the QP problem and the total time expanded as the iteration number increased. Besides, the "Calc Time" is almost unchanged as it only calculates the weighted average of 1000 numbers. In addition, the smallest error appears when the iteration number is 700, but the corresponding number of total samples is 20365, which is much larger than the smallest value 4482. Furthermore, when the iteration number is 1130, the smallest number of samples, 4482, is achieved (here the exact solution to the QP problem is used). However, in this case, the error is 4.411%, which is 13 times bigger than the smallest value 0.339%. The benefit conferred by the constraint aware sampling method in efficiency is due to the following fact. Compared to the importance sampling requiring the biased distribution to be proportional to the original distribution to have small variance, the constraint aware sampling method does not assume the biased distribution to be proportional to the original distribution, leading to a higher efficiency. We set the iteration number to ∞ in the rest experiments for simplicity.

6.3 Impact of the Step Length in the MCMC Sampling Method

In this experiment, we examine the impact of the step length on the performance of our MCMC sampling method by varying the

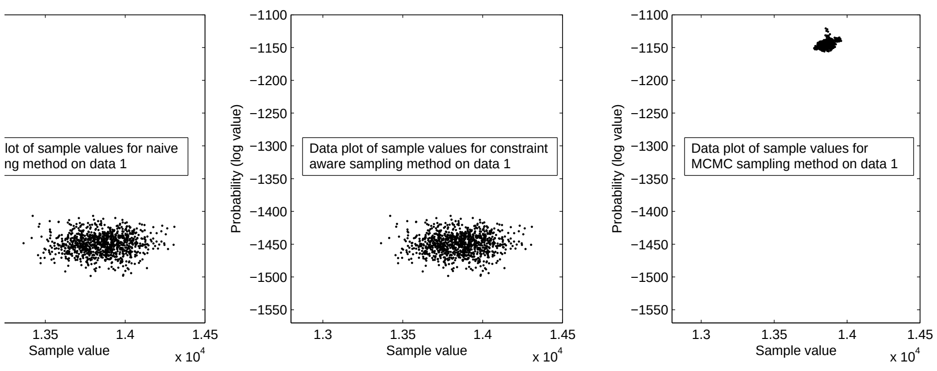

Figure 3: Distribution of valid samples in data 1. The same sampling result is used for the naive sampling method and the constraints aware sampling method in data 1.

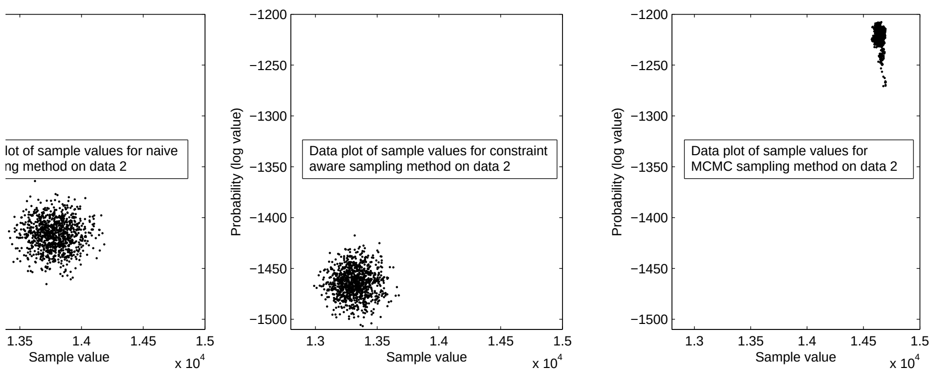

Figure 4: Distribution of valid samples in data 2.

step length from 1, 2, 3 to 50. As shown in Figure 5 and Table 8, when we enlarge the step length, the average probability in the returned sample set increases accordingly, and the sampling error decreases from 6.570% to 1.633%. However, as demonstrated in Table 8, the running time extends correspondingly with increment of the step length. Therefore, to achieve a better tradeoff between accuracy and efficiency, the step length of 10 is suggested and it is how we decided the step length as 10 in our previous experiments.

7. CONCLUSION

In this paper, we studied the problem of querying an uncertain

Itera-tion	Total Sample	Error (%)	QP Time	Sample Time	Calc Time	Total Time
0	161293	14.962	0.40	16.08	49.24	65.71
100	25090	13.816	29.83	2.64	49.22	81.69
200	5664	11.830	55.74	0.69	49.24	105.66
300	6793	10.856	77.65	0.80	49.23	127.68
400	17087	8.763	96.07	1.83	49.19	147.09
500	20066	6.551	111.53	2.11	49.17	162.81
600	20261	3.291	123.74	2.13	49.13	174.99
700	20365	0.339	133.18	2.16	49.13	184.46
800	21013	1.058	140.51	2.19	49.17	191.87
900	20432	4.498	145.96	2.13	49.08	197.16
1000	19450	5.167	149.04	2.03	49.12	200.20
1100	5030	4.846	156.57	0.61	49.16	206.34
1130	4482	4.411	158.65	0.56	49.17	208.38

Table 7: Sampling result of different iteration numbers of quadratic programming on data 2.

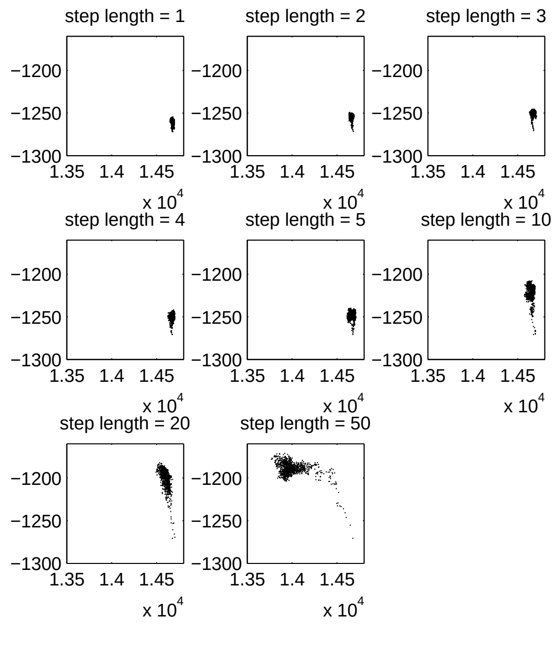

Figure 5: Distribution of valid samples for the MCMC sampling method on data 2 with different step lengths (with 1K valid samples generated, the X-axis is the sample value and the Y-axis is the natural logarithm of the probability of a sample.)

Step Length	Expectation	Error(%)	Time(s)	Total Sample
1	14680.677	6.570	0.718	1541
2	14659.281	6.415	0.813	1742
3	14670.668	6.498	1.016	2210
4	14671.328	6.503	1.125	2448
5	14666.368	6.467	1.406	2974
10	14649.546	6.345	2.438	5110
20	14594.688	5.946	5.422	11639
50	14000.483	1.633	30.86	65196

Table 8: Performance of the MCMC sampling method on data 2 with different step lengths (with 1K valid samples generated).

database with aggregate constraints. In many applications, global statistics of the data (which can be expressed as aggregate constraints) are available, but integrating such constraints with the underlying uncertain data in sampling is often computationally challenging. In this work, we presented two approaches, constraint aware sampling and MCMC sampling, to improve efficiency in sampling possible worlds for query processing. We showed that our approach is effective and efficient for large datasets with a large number of aggregate constraints.

8. REFERENCES

[1] P. Agrawal, O. Benjelloun, A. D. Sarma, C. Hayworth, S. Nabar, T. Sugihara, and J. Widom. Trio: A System for Data, Uncertainty, and Lineage. In *VLDB*, pages 1151–1154, 2006.

[2] P. Andritsos, A. Fuxman, and R. J. Miller. Clean Answers over Dirty Databases: A Probabilistic Approach. In *ICDE*, page 30, 2006.

[3] L. Antova, T. Jansen, C. Koch, and D. Olteanu. Fast and Simple Relational Processing of Uncertain Data. In *ICDE*, pages 983–992, 2008.

[4] L. Antova, C. Koch, and D. Olteanu. Query Language Support for Incomplete Information in the MayBMS System. In *VLDB*, pages 1422–1425, 2007.

[5] S. Chaudhuri, A. D. Sarma, V. Ganti, and R. Kaushik. Leveraging Aggregate Constraints for Deduplication. In *SIGMOD Conference*, pages 437–448, 2007.

[6] H. Chen, W.-S. Ku, and H. Wang. Cleansing uncertain databases leveraging aggregate constraints. In *2nd Workshop on Management and Mining of Uncertain Data (MOUND)*, 2010.

[7] H. Chen, W.-S. Ku, H. Wang, and M.-T. Sun. Leveraging Spatio-temporal Redundancy for RFID Data Cleansing. In *SIGMOD Conference*, pages 51–62, 2010.

[8] R. Cheng, S. Singh, and S. Prabhakar. U-DBMS: A Database System for Managing Constantly-evolving Data. In *VLDB*, pages 1271–1274, 2005.

[9] N. Dalvi and D. Suciu. Efficient Query Evaluation on Probabilistic Databases. *The VLDB Journal*, 16(4):523–544, 2007.

[10] P. Gill, W. Murray, M. Saunders, and M. Wright. Procedures for Optimization Problems with a Mixture of Bounds and General Linear Constraints. *ACM Transactions on Mathematical Software (TOMS)*, 10(3):282–298, 1984.

[11] P. Gill, W. Murray, and M. Wright. Numerical Linear Algebra and Optimization. 1991.

[12] C. P. Gomes, W. J. van Hoeve, A. Sabharwal, and B. Selman. Counting CSP Solutions Using Generalized XOR Constraints. In *AAAI*, pages 204–209, 2007.

[13] D. Gyllstrom, E. Wu, H.-J. Chae, Y. Diao, P. Stahlberg, and G. Anderson. SASE: Complex Event Processing over Streams. In *CIDR*, pages 407–411, 2007.

[14] R. Jampani, F. Xu, M. Wu, L. L. Perez, C. Jermaine, and P. J. Haas. MCDB: A Monte Carlo Approach to Managing Uncertain Data. In *SIGMOD*, pages 687–700, 2008.

[15] R. Jin, L. Liu, B. Ding, and H. Wang. Distance constraint reachability computation in uncertain graphs. *PVLDB*, 2011.

[16] N. Kitchen and A. Kuehlmann. A Markov Chain Monte Carlo Sampler for Mixed Boolean/Integer Constraints. In *CAV*, pages 446–461, 2009.

[17] C. Koch and D. Olteanu. Conditioning Probabilistic Databases. *PVLDB*, 1(1):313–325, 2008.

[18] M. Kozlov, S. Tarasov, and L. Hacijan. Polynomial Solvability of Convex Quadratic Programming. In *Soviet Mathematics Doklady*, volume 20, pages 1108–1111, 1979.

[19] T. Lee, Z. Wang, H. Wang, and S. Hwang. Web scale entity resolution using relational evidence. Technical Report MSR-TR-2011-30, Microsoft Research, 2011.

[20] E. Michelakis, R. Krishnamurthy, P. J. Haas, and S. Vaithyanathan. Uncertainty Management in Rule-based Information Extraction Systems. In *SIGMOD Conference*, pages 101–114, 2009.

[21] C. Papadimitriou. On Selecting a Satisfying Truth Assignment. In *FOCS*, pages 163–169, 1991.

[22] M. Potamias, F. Bonchi, A. Gionis, and G. Kollios. K-nearest neighbors in uncertain graphs. *PVLDB*, 2010.

[23] C. Ré and D. Suciu. The trichotomy of HAVING queries on a probabilistic database. *The VLDB Journal*, 18(5):1091–1116, 2009.

[24] B. Selman, H. Kautz, and B. Cohen. Local Search Strategies for Satisfiability Testing. *DIMACS Series*, 26:521–532, 1996.

[25] B. Selman and H. A. Kautz. Domain-Independent Extensions to GSAT: Solving Large Structured Satisfiability Problems. In *IJCAI*, pages 290–295, 1993.

[26] B. Selman, H. J. Levesque, and D. G. Mitchell. A New Method for Solving Hard Satisfiability Problems. In *AAAI*, pages 440–446, 1992.

[27] P. Sen and A. Deshpande. Representing and Querying Correlated Tuples in Probabilistic Databases. In *ICDE*, pages 596–605, 2007.

[28] W. Shen, X. Li, and A. Doan. Constraint-Based Entity Matching. In *AAAI*, pages 862–867, 2005.

[29] W. Spears. Simulated Annealing for Hard Satisfiability Problems. *DIMACS Series*, 26:533–558, 1996.

[30] E. Tsang. *Foundations of Constraint Satisfaction*. Academic New York, 1993.

[31] J. Xie, J. Yang, Y. Chen, H. Wang, and P. S. Yu. A Sampling-Based Approach to Information Recovery. In *ICDE*, pages 476–485, 2008.

[32] J. Zhu, J. Wang, I. J. Cox, and M. J. Taylor. Risky Business: Modeling and Exploiting Uncertainty in Information Retrieval. In *SIGIR*, pages 99–106, 2009.