Cleansing Uncertain Databases Leveraging Aggregate Constraints

Haiquan Chen[§] Wei-Shinn Ku[§] Haixun Wang[†]
[§]Dept. of Computer Science and Software Engineering, Auburn University, USA
[†]Microsoft Research Asia, Beijing, China
{chenhai,weishinn}@auburn.edu, haixunw@microsoft.com

***Abstract*— Emerging uncertain database applications often involve the cleansing (conditioning) of uncertain databases using additional information as new evidence for reducing the uncertainty. However, past researches on conditioning probabilistic databases, unfortunately, only focus on functional dependency. In real world applications, most additional information on uncertain data sets can be acquired in the form of aggregate constraints (e.g., the aggregate results are published online for various statistical purposes). Therefore, if these aggregate constraints can be taken into account, uncertainty in data sets can be largely reduced. However, finding a practical method to exploit aggregate constraints to decrease uncertainty is a very challenging problem. In this paper, we present three approaches to cleanse (condition) uncertain databases by employing aggregate constraints. Because the problem is NP-hard, we focus on the two approximation strategies by modeling the problem as a nonlinear optimization problem and then utilizing Simulated Annealing (SA) and Evolutionary Algorithm (EA) to sample from the entire solution space of possible worlds. In order to favor those possible worlds holding higher probabilities and satisfying all the constraints at the same time, we define Satisfaction Degree Functions (SDF) and then construct the objective function accordingly. Subsequently, based on the sample result, we remove duplicates, re-normalize the probabilities of all the qualified possible worlds, and derive the posterior probabilistic database. Our experiments verify the efficiency and effectiveness of our algorithms and show that our approximate approaches scale well to large-sized databases.**

I. Introduction

Uncertain and probabilistic database systems have numerous applications such as information retrieval [1], data cleaning [2], computational science [3], decision support and diagnosis systems, sensor and RFID data processing, etc. However, we need techniques beyond the mere retrieval of tuples and their confidence to enhance the power of query languages for supporting advanced applications. For example, most of the existing techniques for processing and managing large uncertain data sets only focus on how to query over a given uncertain database statically [2], [4], [5], [6], [7]. If the uncertain database is inherently highly inaccurate, there will be no chance that the retrieved results are precise. Therefore, as an essential operation, data cleansing is required to reduce uncertainty for subsequent query processing. In this paper, we try to cleanse uncertain databases by adding aggregate constraints as the new *evidence* to refine a prior probabilistic database to a posterior probabilistic database. In other words, we apply a set of aggregate constraints to an uncertain database and materialize the cleaned, less uncertain database. This procedure can be seen as knowledge compilation and have numerous applications in reality.

Employee	Salary (unit:K)	Confidence
Alice	70	0.4
Alice	20	0.4
Alice	90	0.1
Alice	30	0.1
Bob	30	0.7
Bob	80	0.2
Bob	90	0.1
Charles	10	0.3
Charles	20	0.6
Charles	90	0.1

TABLE I
An illustration of the employee database.

Consider the example of an uncertain database of employees and their respective salaries as shown in Table I. Uncertainty exists for the salary of each employee. The alternative distributions of the three tuples can be represented as Equation 1, Equation 2 and Equation 3, respectively.

$$p_x(x) = \begin{cases} 0.4, x = 20 \\ 0.1, x = 30 \\ 0.4, x = 70 \\ 0.1, x = 90 \end{cases} \quad (1)$$

$$p_y(y) = \begin{cases} 0.7, y = 30 \\ 0.2, y = 80 \\ 0.1, y = 90 \end{cases} \quad (2)$$

$$p_z(z) = \begin{cases} 0.3, z = 10 \\ 0.6, z = 20 \\ 0.1, z = 90 \end{cases} \quad (3)$$

Suppose we have acquired a new evidence of the above uncertain database that the sum of the salaries for all the employees is in the range of [50K, 70K]. Thus some of the possible worlds in the above uncertain database (can be treated as a prior probabilistic database) should be removed because they do not satisfy this constraint. For example, the possible world $< x = 70K, y = 30K, z = 20K >$ should be discarded because the sum is equal to 120K which is over the range.

Consequently, we need to retrieve all the qualified possible worlds and re-normalize their probabilities to make them sum up to 1 again. The outcome is the posterior probabilistic database with less uncertainty. In particular, if the aggregate constraint is inconsistent with the given uncertain database, all possible worlds should be deleted.

A. Challenges

The challenges of the problem are as follows:

- The cleansing operation on uncertain databases leveraging aggregate constraints is at least as hard as exact confidence computation, which is proven to be an NP-hard problem [3], [8], [9]. Therefore, we need to utilize approximation mechanisms to solve the problem efficiently and effectively.
- Due to the existence of constraints, a possible world with a higher probability in the prior probabilistic database does not necessarily have a higher probability in the posterior probabilistic database.
- A possible world with a higher probability does not guarantee that it will satisfy all the constraints. On the contrary, a possible world satisfying all the constraints may have a negligible or trivial probability. Therefore, we need to develop techniques to efficiently retrieve the possible worlds holding higher probabilities and satisfying all the constraints at the same time.

B. Overview of Our Approach

In this paper, we propose an original solution of cleansing uncertain databases based on aggregate constraints. Our approaches enable the updating of an uncertain database with additional aggregate knowledge to achieve higher precision and accuracy. In our exact approach, we construct a generation tree of possible worlds and make use of the branch-and-bound technique to prune the tree. Because the cleansing operation is an NP-hard problem, we also propose two approximate strategies by modeling the problem as a nonlinear optimization problem and utilizing simulated annealing and evolutionary algorithm to sample from the entire solution space of possible worlds. The objective function is constructed by using Satisfaction Degree Functions (SDF). Subsequently, based on the sample result, we remove duplicates, re-normalize the probabilities of all the qualified possible worlds, and compute the posterior probabilistic database.

C. Our Contributions

The contributions of our study are as follows:

- To the best of our knowledge, we propose the first solution to cleanse uncertain databases based on aggregate constraints.
- In our exact approach, we construct a generation tree of possible worlds and then employ the branch-and-bound technique to prune the tree to efficiently enumerate all the qualified possible worlds.
- In our approximate approaches, we model the cleansing operation as a nonlinear optimization problem and employ simulated annealing and evolutionary algorithm to sample from the entire solution space of possible worlds.
- We conduct extensive simulations to validate the efficiency and effectiveness of our approximate algorithms and show that they scale well to large-sized databases.

D. Paper Organization

The rest of this paper is organized as follows. The uncertainty representation model and aggregate constraints are formalized in Section II. In Section III, we illustrate our exact approach which employs the branch-and-bound technique to prune the generation tree of possible worlds. Two approximate approaches based on sampling from the entire solution space of possible worlds are presented in Section IV. In Section V, we remove duplicates, re-normalize the probabilities of all the qualified possible worlds, and at last compute the posterior probabilistic database. The experimental validation of our solutions is presented in Section VI. Section VII surveys the related work. Finally, Section VIII concludes the paper.

II. Preliminary

A. Uncertainty Representation

For the uncertainty representation in database systems, *extended relational model* [10], [11], [12], [13], [3] is proposed where the classical relational model is augmented with attribute-level or tuple-level confidence values. In this paper, we define sets of possible worlds following the model of U-relational databases [14]. *U-relations* which can capture most representation formalism for uncertain and probabilistic data was proposed recently in database research literatures, such as Trio [14], MayBMS [12], [15] and MystiQ [3]. In our model, an uncertain relation is a collection of tuples. Each tuple is comprised of one or more mutually-exclusive alternatives. Each alternative is an instantiation of a tuple and holds an associated probabilistic confidence value in the range of $[0, 1]$, i.e., each tuple is associated with a probabilistic mass function. Besides, if a tuple has an infinite number of alternatives, then we can associate a probabilistic density function to the tuple. For each tuple, the confidence values of all the alternatives sum to 1. On the other hand, a *possible world* is a relation generated by selecting exactly one alternative for each tuple. Thus, the total number of possible worlds for an uncertain relation is the product of the number of alternatives of all the tuples. In addition, each possible world has an associated probability, which is the product of the probabilistic confidence values of all the selected alternatives. Therefore, uncertain relations are finite sets of possible worlds with probability weights. Notice that the probability weights of all the possible worlds sum to 1. Consequently, an uncertain relation can be interpreted as a probability distribution over all the possible worlds.

Formally, an uncertain relation is a finite set of structures $R = < {A_1}^1, ..., {A_k}^1, p_1 >, ..., < {A_1}^n, ..., {A_k}^n, p_n >$ such that $\sum_i p_i = 1$, where ${A_k}^i$ is an alternative for tuple T_k in the possible world $pw_i =< {A_1}^i, ..., {A_k}^i >$. p_i is the probability

of pw_i and $0 < p_i \leq 1$. Each tuple T_i is a set of its alternatives, which can be denoted as $A_1, ..., A_m$. Besides, each alternative A_m is a structure of $< x, conf(x) >$, where x is a value and $conf(x)$ is the probabilistic confidence associated to x. In particular, we have $p_i = \prod_{x \in pw_i} conf(x)$ (i.e., tuples are assumed to be independent of each other).

B. Aggregate Constraints

As discussed before, our solution of cleansing uncertain databases takes advantage of aggregate constraints which can be obtained from various data sources, e.g., aggregate results are often published online for statistical purposes. In this paper, we focus on the four SQL aggregate operations: SUM, MAX, MIN, and AVG. We denote an aggregate constraint C_i with the following form: $< C_i = R, T, a, b >$ where R is the subset of data (tuples) that the aggregate constraint is applied on, T is the type of aggregate, and a and b are the lower and upper bounds of the constraint, respectively.

Definition Given a set of aggregate constraints, denoted as C, a *possible world* is defined to be qualified with respect to C if this possible world satisfies every constraint in C.

III. Exact Approach: Brand-and-bound

A. The Generation Tree of Possible Worlds

In our model, uncertain databases are finite sets of possible worlds with probability weights. Thus, if we denote each tuple as a *random variable*, each alternative of a tuple can be considered as an assignment to the corresponding random variable. Consequently, a possible world can be materialized if we assign values to all the random variables (tuples). As shown in Figure 1, all the possible worlds for a specific uncertain database can be enumerated by a generation tree of possible worlds. Figure 1 illustrates the generation tree of possible worlds for the employee database, where all the possible worlds are enumerated.

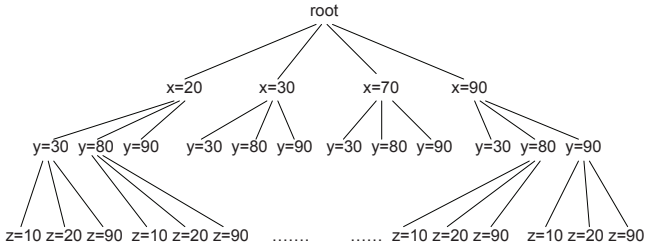


Fig. 1. The generation tree of possible worlds.

B. Traverse the Generation Tree by Branch-and-bound

In order to decrease the size of the generation tree, we use the branch-and-bound technique to prune the entire search space according to the given aggregate constraints.

1) Sorting and Bounding Alternatives: We first sort and bound all the alternatives for each tuple for the subsequent branch-and-bound process. We enumerate all the alternatives for each tuple and arrange them from left to right in the generation tree by ascending order.

2) Branch and Bound: Branch-and-bound [16] is a strategy developed for solving discrete and combinatorial optimization problem. With the branch-and-bound approach, in the generation tree of possible worlds, at any node n, if some of n's descendants cannot generate a qualified possible world, the branches of those descendants can be pruned. Consequently, if enough branches of the generation tree can be pruned, the complexity of the problem of traversing the generation tree can be dramatically reduced. Take the generation tree in Figure 1 as an example. When the branch-and-bound approach is applied, the actual search space for the constraint $50K \leq$ SUM $\leq 70K$ can represented as Figure 2. Since the generation tree grows exponentially with a large-scale database, we present the exact approach mainly for demonstrating the concept. We focus on the following two approximate approaches in this paper.

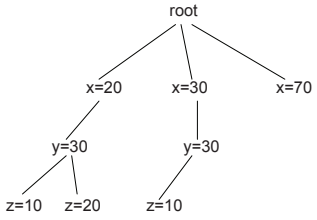


Fig. 2. The actual search space by applying branch-and-bound strategy with the aggregate constraint $50K \leq$ SUM $\leq 70K$.

IV. Approximate Approaches: Nonlinear Optimization Problem

In our approximate approaches, we investigate the problem of cleansing uncertain databases based on aggregate constraints from the nonlinear optimization problem's perspective. At first, we define the satisfaction degree functions to describe the extent to which a specific possible world satisfies aggregate constraints. Then we derive the objective function by integrating satisfaction degree functions with the probabilities of possible worlds. By maximizing the objective function using simulated annealing and evolutionary algorithm, we collect those possible worlds with higher values of the objective function. In this paper, we consider the four SQL aggregate operations: SUM, MAX, MIN, and AVG.

A. Satisfaction Degree Functions of Constraints

For each constraint C_i, we define the satisfaction degree function SDF_{C_i} to evaluate the extent that an aggregate constraint is satisfied. For example, assume that a SUM aggregate constraint has a satisfying range of [a,b] (if the aggregate is MAX, MIN or AVG, the procedure is similar). Although any value within this range is qualified, in order to describe the

extent to which a possible world satisfies this sum aggregate, we assume that the function has the highest satisfaction degree when the value is as $\frac{a+b}{2}$. On the other hand, the satisfaction degree is extremely low for any value $\ll a$ or $\gg b$.

One possible solution is the exponential function, as shown in Equation 4, where x is the aggregate result based on the possible world, pw, and C_i denotes the constraint $a \leq \text{SUM} \leq b$. If the aggregate value is at $\frac{a+b}{2}$ then $SDF_{C_i}(pw) = 1$. The values goes down when the value deviates from $\frac{a+b}{2}$. Z is a normalization factor, which can be tuned based on the concrete aggregate constraint. Figure 3 illustrates the satisfaction degree function for the sum constraint $300000 < \text{SUM} < 700000$, where $a = 300000$, $b = 700000$ and $Z = 100000$. Specifically, in Figure 3, the red lines represent the lower and upper bounds of the SUM aggregate constraint.

$$SDF_{C_i}(pw) = \exp^{-Z \cdot |x - \frac{a+b}{2}|} \tag{4}$$

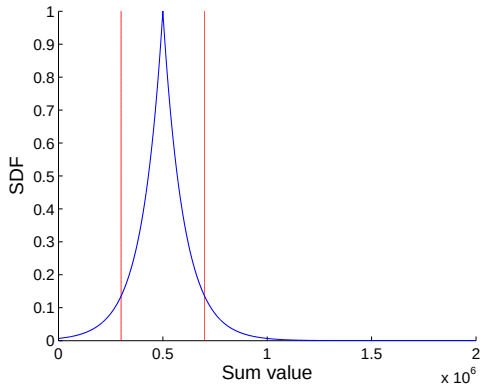


Fig. 3. The satisfaction degree function for the constraint $300000 < \text{SUM} < 700000$.

B. Objective Function

In this subsection, we formalize the objective function by integrating the satisfaction degree functions with the probabilities of possible worlds. The computation of the probability for a possible world can be represented as Equation 5, where pw_i is a possible world, x represents an alternative, $conf(x)$ denotes the confidence of the alternative x, and p_i represents the possibility of pw_i. Based on Equation 5, the probability of a possible world is the multiplication of the confidences of all the alternatives in that possible world. As a result, the Objective Function (OF) of pw_i can be computed as Equation 6, where C denotes the set of aggregate constraints. In the following sections, we employ simulated annealing and evolutionary algorithm to maximize this objective function while keeping track of all the collected samples at the same time.

$$p_i = \prod_{x \in pw_i} conf(x) \tag{5}$$

$$\begin{aligned} OF(pw_i) &= p_i \cdot \prod_{C_i \in C} SDF_{C_i}(pw_i) \\ &= \prod_{x \in pw_i} conf(x) \cdot \prod_{C_i \in C} SDF_{C_i}(pw_i) \end{aligned} \tag{6}$$

C. The Simulated Annealing based Approach

Here we elaborate on our simulated annealing based approach. Simulated annealing [17] is a widely used algorithm for nonlinear optimization problems. It is well-known, however, that SA algorithm does not keep track of any previously accepted states, i.e., it only stores the latest state in the solution space. Therefore, in our design we modified the SA to keep track of each state (possible world) once accepted by SA. From the perspective of sampling, each state in the resulting Markov chain constitutes a possible world.

1) Naive Simulated Annealing: In SA, each step replaces the current solution by a random perturbation mechanism. The probability of making the transition from the current state s to a candidate new state $\acute{s}$ is specified by an acceptance probability function $P(s, \acute{s}, T)$. If the new state $\acute{s}$ is better than the current state s, then the system will move to the new state with a probability of 1. On the contrary, if the new state $\acute{s}$ is worse than the current state s, then the system will move to the new state $\acute{s}$ with a probability of $P(s, \acute{s}, T)$, which will be in the range of [0,1], meaning that the system may move to the new state even when it is worse than the current one. This strategy prevents the method from been trapped in a local minimum.

On the other hand, the temperature is gradually reduced as the search proceeds. Initially, T is set to a high value, and it is decreased at each cooling cycle. When T is close to zero, the probability $P(s, \acute{s}, T)$ tends to zero. To be specific, for a smaller values of T, the system will decreasingly favor moves that go to a worse state. When T becomes 0, the procedure will reduce to the greedy algorithm.

2) The Markov Chain of Possible Worlds: In our improved simulated annealing, we extend the naive simulated annealing to keep track of each state once accepted by the system to maintain a Markov chain of possible worlds, i.e., each accepted state at each move is added to the end of the current state chain. The resulting Markov chain of possible worlds (states) will be used to derive the posterior probabilistic database.

3) The Perturbation Strategy: Because each possible world can be treated as a very high-dimensional vector, whose dimensionality is equal to the number of tuples in the data set, we need to devise an effective scheme to generate the perturbation on the current state (sample). In our design, if we have the current state as $S_k = \langle X_1, X_2, ..., X_n \rangle$, the proposal state $S_k{}'$ will be generated as follows:

- Randomly select a column (tuple) to be changed for the next move;
- Randomly select an alternative for that column;
- Replace the current alternative with the selected alternative for that column.

Algorithm 1 Simulated Annealing Based Approach

1: Load the original uncertain database into the main memory;
2: Initialize the cooling factor;
3: Generate the first state, pw_1, of the Markov chain of possible worlds by randomly selecting one alternative for each tuple;
4: **for** T = Initial temperature to Final temperature **do**
5: **for** $cycle$ = 1 to MAX_NUM **do**
6: Generate a random integer k between 1 and the total number of tuples in the uncertain database;
7: Randomly pick up an alternative alt for the k^{th} tuple;
8: Generate a new possible world, pw_i', by replacing the corresponding alternative for the k^{th} tuple with alt;
9: **if** $OF(pw_i') \geq OF(pw_i)$ **then**
10: Add pw_i' to the end of the Markov chain of possible worlds as the next state;
11: **else**
12: Generate a random number p between 0 and 1;
13: **if** $p \leq \exp\frac{-(OF(pw_i)-OF(pw_i'))}{T}$ **then**
14: Add pw_i' to the end of the Markov chain of possible worlds as the next state;
15: **else**
16: Add pw_i to the end of the Markov chain of possible worlds as the next state;
17: **end if**
18: **end if**
19: $cycle$++;
20: **end for**
21: $T = T \times Cooling\_factor$;
22: **end for**

4) Algorithm Description: The details of the simulated annealing based approach is illustrated in Algorithm 1. In Algorithm 1, line 1 to 3 load the original uncertain database, set up the cooling factor and then generate at random the starting state of the Markov chain of possible worlds. Line 4 to 22 generate the entire Markov chain of possible worlds while the temperature keeps decreasing. Line 5 to 20 produce MAX_NUM states for the Markov chain at each temperature level. Line 6 to 8 generate the next state according to our perturbation strategy. If the value of the objective function of the new state is larger than that of the current state, then store the new state at the end of the Markov chain as shown in line 9 to 10. Otherwise, we compute the acceptance probability and accept the new state as the next state with the computed acceptance probability, as shown in line 12 to 16. Line 19 increments the cycle number. Line 21 cools down the current temperature for constructing the Markov chain at the next temperature level.

D. The Evolutionary Algorithm based Appraoch

Evolutionary algorithms [18] are based on a population of individuals that evolve based on natural selection and mutation. Each individual, representing a candidate solution, competes with its siblings based on the fitness function. Evolutionary algorithms have been proven to be efficient on optimization, modeling, and simulation problems. In our evolutionary algorithm based approach, we treat the objective function, as shown in Equation 6, as the fitness function, and then use the steady-state evolutionary algorithm with elitist selection to gradually evolve the population of solutions to maximize the objective function.

1) Encoding of Individuals and Constraints: To implement the evolutionary algorithm, we need to choose a representation for the individuals (solutions). For our problem, each individual corresponds to a possible world of the uncertain database consisting of alternatives (one alternative per tuple).

2) Fitness Evaluation: In our evolutionary algorithm, the fitness function is defined as the objective function as shown in Equation 6.

3) Evolution Process:

- **Parent selection** During parent selection, individuals that are allowed to mate and create offspring will be selected from the current population. We employ a tournament-based selection, in which two individuals are randomly picked from the population at a time and each individual holds the same probability to be selected.
- **Crossover** Crossover is used to create new individuals in unexplored regions of the search space. It takes two parents and combines them into two children. Here, we opt for a classical one-point crossover operator. One pivot gene is randomly selected and the two parts around it are swapped between the two parents to generate two children.
- **Mutation** Mutation is aimed at exploring the neighborhood of an individual, compared with crossover whose goal is to produce big jumps in the search space. To perform mutation for an individual, we first randomly select a gene (tuple) in that individual, then select at random an alternative for that specific gene, and finally replace the gene with the selected alternative.
- **Survival selection** We employ an elitist selection strategy in our evolutionary algorithm. Once we get offspring, we compare them with all the parents and survive the individuals with higher fitness values.

4) Algorithm Description: Our steady-state evolutionary algorithm based approach is formalized in Algorithm 2. Line 1 to 3 load the original uncertain database and initialize the population size and the total number of generations. Then line 4 to 6 generate the first generation of the population at random in the entire solution space. Line 7 to 16 produce the MAX-1 generations of the population. Line 8 to 11 choose parents by a tournament-based selection and conduct crossover and mutation on them to generate offspring. Then we compute the values of the objective function for all the parents and the children and only survive the top S individuals for the next generation as shown in line 12 to 14. Line 15 increments the generation number.

V. Posterior Probabilistic Databases

By using simulated annealing or evolutionary algorithm to walk on the solution space randomly, we are able to obtain a collection of samples (possible worlds). Afterwards, given all the samples, we remove duplicates from them, re-normalize the probabilities of all the qualified possible worlds, and derive the posterior probabilistic database at last.

Algorithm 2 Steady-state Evolutionary Alg. Based Approach

1: Load the original uncertain database into the main memory;
2: Initialize the population size, S;
3: Initialize the total number of generations, MAX;
4: **for** i = 1 to S **do**
5: Generate an individual (possible world), pw_i, by randomly selecting an alternative for each tuple in the uncertain database;
6: **end for**
7: **for** $generation$ = 2 to MAX **do**
8: **for** k = 1 to $\frac{S}{2}$ **do**
9: Randomly select two parents from the current generation, pw_m and pw_n;
10: Generate two children, pw'_m and pw'_n by crossover and mutation on pw_m and pw_n;
11: **end for**
12: Compute $OF(pw)$ for all the parents and $OF(pw')$ for all the children, respectively;
13: Survive the top S individuals with the highest OF value;
14: Store the selected S individuals as the population for the next generation;
15: $generation$++;
16: **end for**

A. Confidences Re-normalization

Suppose we have a qualified possible world, pw_m, and its probability in the prior probabilistic database can be denoted as $p(pw_m)$. To compute the probability of pw_m in the posterior probabilistic database, which is denoted as $p'(pw_m)$, we can obtain

$$p'(pw_m) = \frac{p(pw_m)}{\sum_{\forall pw \in A} p(pw)} \approx \frac{p(pw_m)}{\sum_{\forall pw \in S} p(pw)} \quad (7)$$

where A denotes the set of all the possible worlds which satisfy all the constraints and S denotes the set of all the sampled possible worlds which satisfy all the constraints, respectively.

Taking the employee database as an example, suppose we have a SUM aggregate constraint $50K \leq \text{SUM} \leq 70K$. Then there are three qualified possible worlds, $< x = 20, y = 30, z = 10 >$, $< x = 20, y = 30, z = 20 >$ and $< x = 30, y = 30, z = 10 >$. Their probabilities in the prior probabilistic database are $0.4 \times 0.7 \times 0.3 = 0.084$, $0.4 \times 0.7 \times 0.6 = 0.168$ and $0.1 \times 0.7 \times 0.3 = 0.021$, respectively. After re-normalization, their probabilities in the posterior probabilistic database are $\frac{0.084}{0.273} = 0.308$, $\frac{0.168}{0.273} = 0.615$ and $\frac{0.021}{0.273} = 0.077$.

Employee	Salary (unit:K)	Confidence
Alice	20	0.923
Alice	30	0.077
Bob	30	1
Charles	10	0.385
Charles	20	0.615

TABLE II
THE CLEANSED (POSTERIOR PROBABILISTIC) EMPLOYEE AND SALARY DATABASE.

B. Calculation of the Marginal Distributions of Tuples

After we have obtained the possible worlds in the posterior probabilistic database with their probabilities, for each alternative a of each tuple, we summarize the probabilities of each possible world where a exists to compute the confidence value of a in the posterior probabilistic database, i.e., compute the marginal distributions for each tuple. Consequently, we can obtain the posterior probabilistic database. Using the employee database as an example, we can obtain its posterior probabilistic database as shown in Table II. Compared with Table I, the database in Table II contains much less uncertainty. In particular, the tuple for the employee "Bob" has been cleansed from a non-deterministic one to a deterministic one.

VI. EXPERIMENTAL VALIDATION

We implemented our simulated annealing and evolutionary algorithm based approaches in GNU C++ to evaluate their performances. The experimental results are reported in this section. We varied the cooling factor, the number of generations, and the number of tuples to obtain their effects on the running time. We used synthetic data sets for our experiments where each tuple is assumed to have two alternatives and each alternative is a random integer within the range of $[0, 100]$. For each experiment, to make the aggregate constraints consistent with our uncertain data sets, we assumed that the sum aggregate constraint can be denoted as $30 \times N \leq$ SUM $\leq 70 \times N$, where N represents the number of tuples in the corresponding experiment. In addition, to show the scalability of our approaches, we varied the number of tuples in the synthetic data sets from $5K$ to $1M$. For the experiments on simulated annealing, we varied the number of the state accepted at each temperature level from 100 (SA (100)), 200 (SA (200)) to 500 (SA (500)). For the experiments on the evolutionary algorithm, we changed the population size from 100 (EA (100)), 200 (EA (200)) to 500 (EA (500)). We ran each experiment 100 times to obtain the average values. All the experiments were conducted on a computer with an Intel Core2 Quad CPU (Q9400 2.66GHz).

A. The Simulated Annealing based Approach

1) The Impact of the Cooling Factor: Here we evaluated the performance of SA (100), SA (200) and SA (500) by varying the cooling factor. We assumed the number of tuples to be 100K. As Figure 4(a) demonstrates, when we raised the cooling factor, all the running time of SA (100), SA (200) and SA (500) kept increasing. For example, the running time for SA (500) was 29.820 seconds at the cooling factor of 0.8 while the running time for SA (500) jumped to 60.779 seconds at the cooling factor of 0.9. The reason is that the higher the cooling factor is, the more slowly the algorithm cools down. In other words, with a higher cooling factor, the algorithm will construct a longer Markov chain of possible worlds in the entire solution space .

2) The Impact of the Number of Tuples: In this experiment, we varied the number of tuples to investigate its effects on the running time of SA (100), SA (200) and SA (500). We set the cooling factor as 0.9. As demonstrated in Figure 4(b), with the increase of the number of tuples, the running time of SA (100), SA (200) and SA (500) was increasing near linearly. Particularly, when the number of tuples is 50K, the running time for SA (500) was 30.469 seconds. When the number of tuples increased to 1M, the running time for SA (500) enlarged to 718.486 seconds.

B. The Evolutionary Algorithm based Approach

1) The Impact of the Number of Generations: We studied the performance of EA (100), EA (200) and EA (500) on the running time by varying the number of generations from 100 to 2000. For each experiment, we assumed the number of tuples to be 10K. Figure 5(a) illustrates the results. With the enlargement of the number of generations, all the running time for EA (100), EA (200) and EA (500) elevated accordingly. For instance, when the number of generation was 500, the running time for EA (500) was 73.989 seconds. On the contrary, when the number of generation raised to 2000, the running time for EA (500) increased to 288.022 seconds. The reason is that when the number of generations increases, the algorithm needs more time to evolve through all the generations.

2) The Impact of the Number of Tuples: Next, we increased the number of tuples from 5K to 500K to investigate the performance of EA (100), EA (200) and EA (500) on the running time. We assumed that the number of generations was 100. As demonstrated in Figure 5(b), when the number of tuples increased, all the running time extended dramatically. In particular, the running time for EA (500) was 84.987 seconds when the number of tuples was 50K. On the other hand, the running time for EA (500) increased to 885.003 seconds when the number of tuples was 500K. The reason is that with the increase of the number of tuples, the chromosomes become longer. Then more time is needed to finish each phase of the evolutionary algorithm (i.e., parents selection, crossover, mutation and survival selection).

VII. Related Work and Background Knowledge

In this section we review previous works related to our approaches of cleansing uncertain databases.

A. Uncertainty Model

Here we introduce the relation models which have been proposed to handle uncertainty in database systems.

1) Independent Tuples Model: The *independent tuples model* [10], [11], [12], [13], [3] makes simplistic and restrictive assumption that each tuple is completely independent and thus, the resulting possible world interpretation lends highly intuitive and precise semantics for query evaluation over uncertain databases. This independence assumption, however, may be problematic because many applications naturally produce correlated tuples. Besides, data generated by sensor networks is intrinsically highly correlated. Therefore, ignoring such correlations can result in highly inaccurate query results.

2) Correlated Tuples Model: By using a probabilistic graphical model, Sen et al. [19] present a framework to incorporate tuple correlations. However, their proposed method requires that dependencies and independencies between events and conditional probabilities are known. In addition, the technique based on graphical models tends to be highly inefficient when an uncertain data set is obtained by queries on U-relations because the corresponding graphical model will be flat and lead to a high width. Furthermore, graphical models are not easily integrated in the current query evaluation mechanisms developed for the table based representation.

B. Conditioning Uncertain Databases

Additional constraints may affect the distribution of possible worlds (and finally, the query results) on uncertain databases. Koch et al. [8] define this problem as *conditioning*. In *conditioning*, all possible worlds which do not satisfy given constraints will be deleted and queries are evaluated only over the remaining possible worlds. Unfortunately, past research on conditioning only focus on functional dependency. However, within the database field, most of *evidences* exist in the form of aggregate results, e.g., related aggregate results are often published by the third party for statistical purposes. Therefore, in our paper, we propose three approaches to cleansing (updating) uncertain databases based on aggregate constraints.

C. Monte Carlo Approximation Techniques

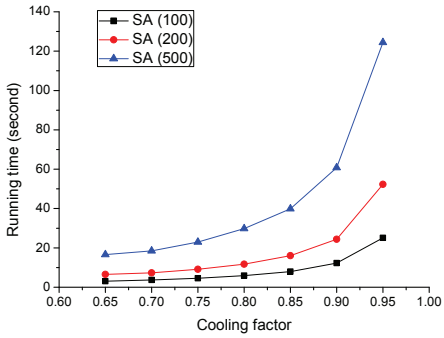
Karp et al. [20] present the original work on confidence computation using monte carlo approximation techniques. Our work can be considered as an attempt to follow this methodology within the database field. In particular, our approximate approaches employ a monte carlo walk to sample possible worlds from the entire solution space via simulated annealing and evolutionary algorithm.

VIII. Conclusions

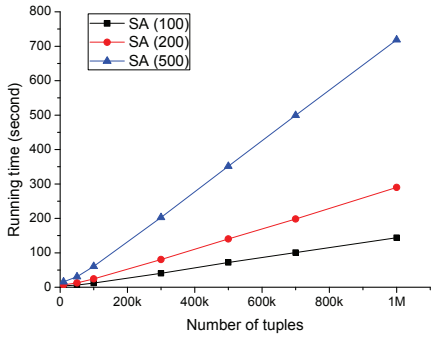
In this paper, we present three approaches to cleanse uncertain databases based on aggregate constraints. Because the problem is NP-hard, we focus on the two approximate approaches, which utilize simulated annealing and evolutionary algorithm to maximize objective functions. Subsequently, based on the samples, we remove duplicates, re-normalize the probabilities of all the qualified possible worlds, and compute the posterior probabilistic database. Our experiment results verify the efficiency and effectiveness of our algorithms and show that our approaches scale well to large-sized databases.

IX. Acknowledgments

This research has been funded in part by the National Science Foundation grants CNS-0831502 (CT), CNS-0855251 (CRI).

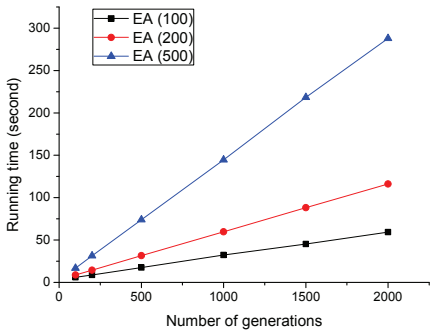


(a) Running time v.s. cooling factor.

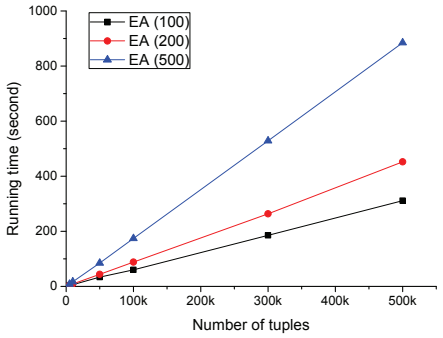


(b) Running time v.s. number of tuples.

Fig. 4. The simulated annealing based approach.



(a) Running time v.s. number of generations.



(b) Running time v.s. number of tuples.

Fig. 5. The evolutionary algorithm based approach.

REFERENCES

[1] N. Fuhr and T. Rölleke, "A probabilistic Relational Algebra for the Integration of Information Retrieval and Database Systems," *ACM Trans. Inf. Syst.*, vol. 15, no. 1, pp. 32–66, 1997.

[2] L. Antova, C. Koch, and D. Olteanu, "10^{10^6} Worlds and Beyond: Efficient Representation and Processing of Incomplete Information," in *ICDE*, 2007, pp. 606–615.

[3] N. N. Dalvi and D. Suciu, "Efficient Query Evaluation on Probabilistic Databases," *VLDB J.*, vol. 16, no. 4, pp. 523–544, 2007.

[4] D. Olteanu, C. Koch, and L. Antova, "World-set Decompositions: Expressiveness and Efficient Algorithms," *Theor. Comput. Sci.*, vol. 403, no. 2-3, pp. 265–284, 2008.

[5] L. Antova, T. Jansen, C. Koch, and D. Olteanu, "Fast and Simple Relational Processing of Uncertain Data," *CoRR*, vol. abs/0707.1644, 2007.

[6] D. Olteanu and J. Huang, "Using OBDDs for Efficient Query Evaluation on Probabilistic databases," in *SUM*, 2008, pp. 326–340.

[7] D. Olteanu, J. Huang, and C. Koch, "SPROUT: Lazy vs. Eager Query Plans for Tuple-Independent Probabilistic Databases," in *ICDE*, 2009, pp. 640–651.

[8] C. Koch and D. Olteanu, "Conditioning Probabilistic Databases," *PVLDB*, vol. 1, no. 1, pp. 313–325, 2008.

[9] N. N. Dalvi and D. Suciu, "Management of Probabilistic Data: Foundations and Challenges," in *PODS*, 2007, pp. 1–12.

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

[11] P. Andritsos, A. Fuxman, and R. J. Miller, "Clean Answers over Dirty Databases: A Probabilistic Approach," in *ICDE*, 2006, p. 30.

[12] L. Antova, C. Koch, and D. Olteanu, "MayBMS: Managing Incomplete Information with Probabilistic World-Set Decompositions," in *ICDE*, 2007, pp. 1479–1480.

[13] R. Cheng, S. Singh, and S. Prabhakar, "U-DBMS: A Database System for Managing Constantly-Evolving Data," in *VLDB*, 2005, pp. 1271–1274.

[14] O. Benjelloun, A. D. Sarma, A. Y. Halevy, and J. Widom, "ULDBs: Databases with Uncertainty and Lineage," in *VLDB*, 2006, pp. 953–964.

[15] J. Huang, L. Antova, C. Koch, and D. Olteanu, "MayBMS: A Probabilistic Database Management System," in *SIGMOD Conference*, 2009, pp. 1071–1074.

[16] D. DeHaan and F. W. Tompa, "Optimal Top-down Join Enumeration," in *SIGMOD Conference*, 2007, pp. 785–796.

[17] V. Granville, M. Krivánek, and J.-P. Rasson, "Simulated annealing: A proof of convergence," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 16, no. 6, pp. 652–656, 1994.

[18] X. Zou, Y. Chen, M. Liu, and L. Kang, "A new evolutionary algorithm for solving many-objective optimization problems," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 38, no. 5, pp. 1402–1412, 2008.

[19] P. Sen and A. Deshpande, "Representing and Querying Correlated Tuples in Probabilistic Databases," in *ICDE*, 2007, pp. 596–605.

[20] R. M. Karp, M. Luby, and N. Madras, "Monte-Carlo Approximation Algorithms for Enumeration Problems," *J. Algorithms*, vol. 10, no. 3, pp. 429–448, 1989.