# Modeling and Querying Possible Repairs in Duplicate Detection

George Beskales Mohamed A. Soliman Ihab F. Ilyas Shai Ben-David
University of Waterloo
{gbeskale,m2ali,ilyas,shai}@cs.uwaterloo.ca

## ABSTRACT

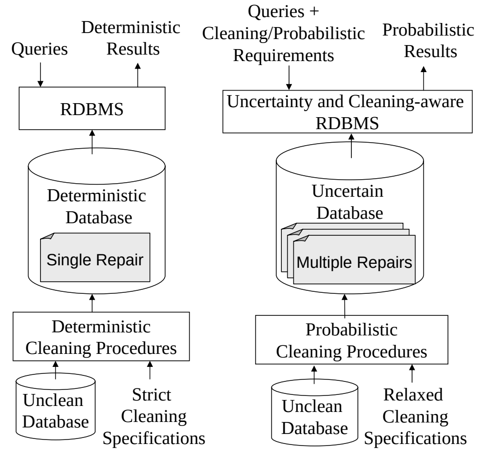
One of the most prominent data quality problems is the existence of duplicate records. Current duplicate elimination procedures usually produce one clean instance (repair) of the input data, by carefully choosing the parameters of the duplicate detection algorithms. Finding the right parameter settings can be hard, and in many cases, perfect settings do not exist. Furthermore, replacing the input dirty data with one possible clean instance may result in unrecoverable errors, for example, identification and merging of possible duplicate records in health care systems.

In this paper, we treat duplicate detection procedures as data processing tasks with uncertain outcomes. We concentrate on a family of duplicate detection algorithms that are based on parameterized clustering. We propose a novel uncertainty model that compactly encodes the space of possible repairs corresponding to different parameter settings. We show how to efficiently support relational queries under our model, and to allow new types of queries on the set of possible repairs. We give an experimental study illustrating the scalability and the efficiency of our techniques in different configurations.

## 1. INTRODUCTION

Data quality is a key requirement for effective data analysis and processing. In many situations, the quality of business and scientific data is impaired by several sources of noise (e.g., heterogeneity in schemas and data formats, imperfection of information extractors, and imprecision of reading devices). Such noise generates many data quality problems (e.g., missing values [14], violated constraints [21], and duplicate records [27, 16]) that impact the effectiveness of many data querying and analysis tasks. Databases that experience such problems are usually referred to as unclean/dirty databases. Data cleaning is a labor intensive process that is intended to repair data errors and anomalies.

Many current commercial tools [1, 2] support the extraction, transformation and loading (ETL) of business (possibly unclean) data into a trust-worthy (cleansed) database. The functionalities offered by ETL tools include data extraction, standardization, record merging, missing values imputation, and many other tasks. We refer to this approach as the one-shot cleaning approach (Figure 1(a) ), where cleaning procedures process the unclean data based on cleaning specifications such as carefully chosen parameter settings of cleaning algorithms, or fixed rules and business logic, to produce a single clean data instance (i.e., a repair for the unclean database).


Queries
Deterministic Results
RDBMS
Deterministic Database
Single Repair
Deterministic Cleaning Procedures
Unclean Database
Strict Cleaning Specifications
Queries + Cleaning/Probabilistic Requirements
Probabilistic Results
Uncertainty and Cleaning-aware RDBMS
Uncertain Database
Multiple Repairs
Probabilistic Cleaning Procedures
Unclean Database
Relaxed Cleaning Specifications

**(a) One-Shot Cleaning** **(b) Probabilistic Cleaning**

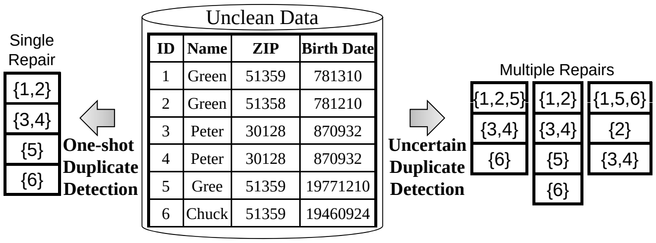
**Figure 1: One-shot Vs. Probabilistic Cleaning**

### 1.1 Motivation and Challenges

Data cleaning often involves uncertainty (e.g., in deciding whether two records are duplicates). Generating a single repair necessitates resolving the uncertain cases deterministically, which is usually done using fixed rules and/or domain experts. This approach may induce errors in the cleansed output. Such errors are unrecoverable once the cleansed output is decoupled from the dirty data and loaded in the database. Moreover, relying on domain experts involves extensive labor work, which is infeasible for large databases. Additionally, maintaining only one clean instance imposes strong dependency between the cleansed data and the used cleaning procedures. Changing these procedures (or their parameter settings) invalidates the cleansed data, and requires re-applying the cleaning procedures.

Maintaining multiple repairs for the dirty data addresses these problems by capturing the uncertainty in the cleaning process and allowing for specifying query-time cleaning requirements. We propose using principles of probabilistic databases to build a cleaning system with such capabilities. We refer to our approach as the probabilistic cleaning approach (Figure 1(b) ), where an uncertainty model is used to describe the possible repairs that represent differ-

*VLDB '09,* August 24-28, 2009, Lyon, France



| ID | Name | ZIP | Birth Date |
|---|---|---|---|
| 1 | Green | 51359 | 781310 |
| 2 | Green | 51358 | 781210 |
| 3 | Peter | 30128 | 870932 |
| 4 | Peter | 30128 | 870932 |
| 5 | Gree | 51359 | 19771210 |
| 6 | Chuck | 51359 | 19460924 |

**Figure 2: One-shot vs. Probabilistic Duplicate Detection**

ent clean data instances. This approach can be useful in various settings. For example, a user may require finding the minimum and maximum possible numbers of distinct entities (e.g., clients) in a dirty database for best-case and worst-case capacity planning. Another example is aggregation queries, where a user may require probabilistically quantified aggregate values (e.g., in the form of confidence intervals), rather than single values, to drive decision making processes based on the clean data with high confidence.

In this paper, we study the problems of modeling and querying possible repairs in the context of duplicate detection, which is the process of detecting records that refer to the same real-world entity. We give the next example to illustrate our approach.

EXAMPLE 1. *Figure 2 shows an input relation representing sample census data that possibly contains duplicate records. Duplicate detection algorithms generate a clustering of records (represented as sets of record ID's in Figure 2), where each cluster is a set of duplicates that are eventually merged into one representative record per cluster.*

In Example 1, the one-shot duplicate detection approach identifies records as either duplicates or non-duplicates based on the given cleaning specifications (e.g., a single threshold on records' similarity). Hence, the result is a single clustering (repair) of the input relation. On the other hand, in the probabilistic duplicate detection approach, this restriction is relaxed to allow for uncertainty in deciding on the true duplicates (e.g., based on multiple similarity thresholds). The result is a set of multiple possible clusterings (repairs) as shown in Figure 2.

Probabilistic modeling of possible repairs is motivated by the fact that errors in similarity measures induce noise in the input of the deduplication algorithms. In fact, it is common to assume a probabilistic model of such noise (e.g., [24]), which induces a probability model over the parameter settings governing the deduplication algorithms. In this paper, our focus is not on providing a new duplicate detection technique with better quality (e.g., in terms of precision and recall), but on providing an approach to model and query the uncertainty in duplicate detection methods with the objective of allowing for greater flexibility in cleaning specifications.

The challenges involved in adopting a probabilistic duplicate detection approach are summarized as follows:

- Generation of Possible Repairs: Uncertainty in duplicate detection yields a space of possible repairs. Efficient generation of such a space, and quantifying the confidence of each repair are key challenges in our problem.

- Succinct Representation of Possible Repairs: The space of possible repairs can be huge as it corresponds to all possible clusterings. Identifying the set of likely repairs and compactly encoding such a set is imperative for efficient storage and processing of the cleansed output.

- Query Processing: Modeling possible repairs allows supporting new query types with online cleaning requirements (e.g., a query that finds record clusters belonging to the majority of possible repairs). Efficient answering of these queries, in addition to conventional relational queries, is challenging as it involves processing multiple repairs.

## 1.2 Contributions

Our key contributions are summarized as follows:

- We introduce an uncertainty model for representing the possible repairs generated by any fixed parameterized clustering algorithm. The model allows specifying cleaning parameters as query predicates by pushing the expensive record clustering operations to an offline construction step.

- We show how to modify hierarchical clustering algorithms to efficiently generate the possible repairs.

- We describe how to evaluate relational queries under our model. We also propose new query types with online cleaning requirements that are not supported by current one-shot cleaning methods.

- We show how to integrate our approach into relational DBMSs to allow storage of possible repairs and performing probabilistic query processing efficiently.

We also conduct an experimental study to evaluate the scalability and efficiency of our techniques under different configurations.

The remainder of this paper is organized as follows. In Section 2, we define the space of possible repairs. An algorithm-dependent uncertainty model is given in Section 3. In Section 4, we discuss how to support relational queries. In Section 5, we discuss implementing our model inside relational DBMSs and show new query types supported by our system. An experimental evaluation is given in Section 6. We discuss related works in Section 7. We conclude the paper with final remarks in Section 8.

## 2. THE SPACE OF POSSIBLE REPAIRS

In this section, we define the space of possible repairs and describe multiple approaches to limit the space size for efficient processing. We start by formally defining a possible repair:

DEFINITION 1. **Duplication Repair**. *Given an unclean relation $R$, a repair $X$ is a set of disjoint record clusters $\{C_1, \ldots, C_m\}$ such that $\bigcup_{i=1}^{m} C_i = R$.* □

A repair $X$ partitions $R$ into disjoint sets of records that cover $R$. By coalescing each set of records in $X$ into a representative record, we obtain a clean (duplicate-free) instance of $R$.

Repairs have clear analogy to the concept of 'possible worlds' in uncertain databases [22, 4, 15], where possible worlds are all possible database instances originating from tuple and/or attribute uncertainty. However, in our settings, the repairs emerge from uncertainty in deciding whether a set of records are duplicates or not.

In general, there are two key problems when dealing with the space of all possible repairs. First, the number of possible repairs can be as large as the number of possible clusterings of $R$, which is exponential in $|R|$ (by correspondence to the problem of set partitioning [7]). Second, quantifying the uncertainty in the space of possible repairs by, for example, imposing a probability distribution on the space of possible repairs, is not clear without understanding the underlying process that generates the repairs.

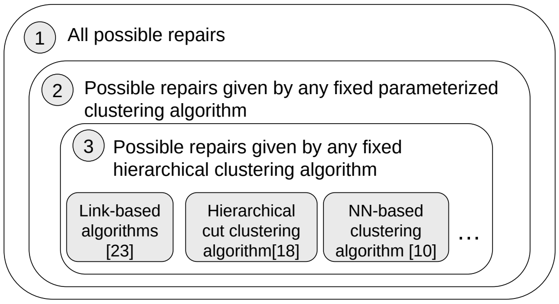

1 All possible repairs
2 Possible repairs given by any fixed parameterized clustering algorithm
3 Possible repairs given by any fixed hierarchical clustering algorithm
Link-based algorithms [23]
Hierarchical cut clustering algorithm[18]
NN-based clustering algorithm [10]
…

**Figure 3:** **Constraining the Space of Possible Repairs**

There are multiple ways to constrain the space of possible repairs including imposing hard constraints to rule out impossible repairs, or filtering the repairs that do not meet specific requirements (e.g., specifying minimum pairwise distance among clustered records). In this paper, we consider the subset of all possible repairs that are generated by any fixed duplicate detection algorithm.

**Algorithm-Dependent Possible Repairs**
Given any parameterized cleaning algorithm, we limit the space of possible repairs to those generated by the algorithm using different parameter settings. This approach has two effects:

1. Limiting the space of possible repairs has a direct effect on the efficiency of query processing algorithms and the space required to store the repairs.
2. By assuming (or learning) a probability distribution on the values of the parameters of the algorithm, we can induce a probability distribution on the space of possible repairs, which allows for a richer set of probabilistic queries (e.g., finding the most probable repair, or finding the probability of clustering two specific records together).

Constraining parameterized algorithms to a specific class of algorithms can further reduce the space complexity and improve the efficiency of query processing algorithms. More specifically, for hierarchical clustering algorithms, the size of the space of possible repairs is linear in the number of records in the unclean relation (we give more details in Section 3.2). Moreover, a hierarchical clustering algorithm can efficiently generate the possible repairs through simple modifications to the algorithm.

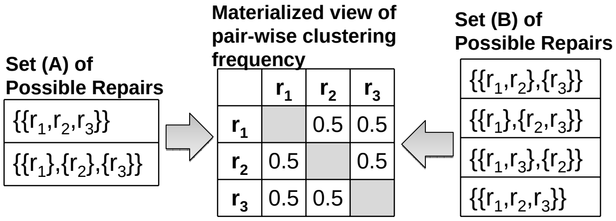
Figure 3 depicts the containment relationship between the space of all possible repairs, and the possible repairs generated by any given parameterized algorithm. Figure 3 also shows examples of hierarchical clustering methods that we discuss in Section 3.2.

## 3. MODELING POSSIBLE REPAIRS

In this Section, we discuss how to represent a set of possible repairs. We focus on modeling the space of possible repairs generated by any fixed clustering algorithm.

There are multiple approaches that can be adopted to model possible repairs. In the following, we present two extremes within the spectrum of possible representations.

- The first model is the triple $(R, \mathcal{A}, \mathcal{P})$, where $R$ denotes the unclean relation, $\mathcal{A}$ denotes a fixed clustering algorithm, and $\mathcal{P}$ denotes a set of possible parameter settings for the algorithm $\mathcal{A}$. This approach is a compact representation that does not materialize any possible repairs, and thus no construction cost is incurred.



Set (A) of Possible Repairs

| |
|---|
| $\{\{r_1,r_2,r_3\}\}$ |
| $\{\{r_1\},\{r_2\},\{r_3\}\}$ |

Materialized view of pair-wise clustering frequency

| | $r_1$ | $r_2$ | $r_3$ |
|---|---|---|---|
| $r_1$ | | 0.5 | 0.5 |
| $r_2$ | 0.5 | | 0.5 |
| $r_3$ | 0.5 | 0.5 | |

Set (B) of Possible Repairs

| |
|---|
| $\{\{r_1,r_2\},\{r_3\}\}$ |
| $\{\{r_1\},\{r_2,r_3\}\}$ |
| $\{\{r_1,r_3\},\{r_2\}\}$ |
| $\{\{r_1,r_2,r_3\}\}$ |

**Figure 4:** **An example model of possible repairs**

- The second model is the set of all possible repairs that can be generated by the algorithm $\mathcal{A}$ using all possible parameter settings $\mathcal{P}$.

Other representations between these two extremes involve (partial) materialization of possible repairs and storing views that possibly aggregate these repairs. For example, a possible representation is to associate each pair of records with the relative frequency of repairs in which both records belong to the same cluster (i.e., declared as duplicates). The problem of finding a suitable view of the possible repairs is analogous to the problem of selecting which materialized views to build in relational databases. Choosing a suitable view depends on several factors such as the cost of materializing the view and the types of queries that can be answered using that view as we illustrate in Example 2.

EXAMPLE 2. *Consider two sets of possible repairs, denoted $A$ and $B$, that involve the base records $\{r_1, r_2, r_3\}$ as shown in Figure 4. The relative frequency of repairs in which two records are clustered together is the same for all pairs of records w.r.t. both sets of repairs. These frequencies are shown in the symmetric matrix in Figure 4.*

The view consisting of the pair-wise clustering frequency depicted in Figure 4 can be used to efficiently answer some queries (e.g., is there any repair in which $r_1$ and $r_2$ are clustered together). However, Example 2 shows that the proposed view is a lossy representation of repairs. That is, this view cannot be used to restore the encoded set of possible repairs. Therefore, such representation cannot be used in answering any given query that is answerable using the set of possible repairs. For example, finding the relative frequency of repairs in which $r_1$, $r_2$ and $r_3$ are clustered together is not possible using the view in Figure 4.

We summarize our proposed desiderata regarding modeling the possible repairs as follows:

- The model should be a *lossless* representation of the possible repairs in order to allow answering queries that require a complete knowledge about these repairs. In other words, we have to ensure that all possible repairs can be restored using the model.
- The model should allow efficient answering of a set of important queries types (e.g., queries frequently encountered in applications).
- The model should provide materialization of the results of costly operations (e.g., clustering procedures) that are required by most queries.
- The model should have small space complexity to allow efficient construction, storage and retrieval of the possible repairs, in addition to efficient query processing.

In Section 3.1, we describe our proposed model that addresses the aforementioned requirements. In Section 3.2, we show how to efficiently obtain the possible repairs for the class of hierarchical clustering algorithms.

## 3.1 Algorithm-Dependent Model

In this section, we introduce a model to compactly encode the space of possible repairs generated by any fixed parameterized duplicate detection algorithm $\mathcal{A}$ that uses a single parameter to determine its output clustering.

We represent possible parameter values of a duplicate detection algorithm $\mathcal{A}$ using a random variable $\tau$. For the sake of a concrete discussion, we assume that $\tau$ is a continuous random variable. However, it is straightforward to adapt our approach to address discrete parameters. We denote by the interval $[\tau^l, \tau^u]$ the possible values of $\tau$, and we denote by $f_\tau$ the probability density function of $\tau$ defined over $[\tau^l, \tau^u]$.

The probability density function $f_\tau$ can be given explicitly based on user's experience, or it can be learned from training data. While learning $f_\tau$ is an interesting problem by itself, we do not study this problem in this paper. Hence, we assume that $f_\tau$ is given. Applying $\mathcal{A}$ to an unclean relation $R$ using a parameter value $t \in [\tau^l, \tau^u]$ generates a possible clustering (i.e., repair) of $R$, denoted $\mathcal{A}(R, t)$. Multiple parameter values may lead to the same clustering.

The set of possible repairs $\mathcal{X}$ is defined as $\{\mathcal{A}(R, t) : t \in [\tau^l, \tau^u]\}$. The set $\mathcal{X}$ defines a probability space created by drawing random parameter values from $[\tau^l, \tau^u]$, based on the density function $f_\tau$, and using the algorithm $\mathcal{A}$ to generate the possible repairs corresponding to these values. The probability of a specific repair $X \in \mathcal{X}$, denoted $\Pr(X)$, is derived as follows:

$$\Pr(X) = \int_{\tau^l}^{\tau^u} f_\tau(t) \cdot h(t, X)\, dt \qquad (1)$$

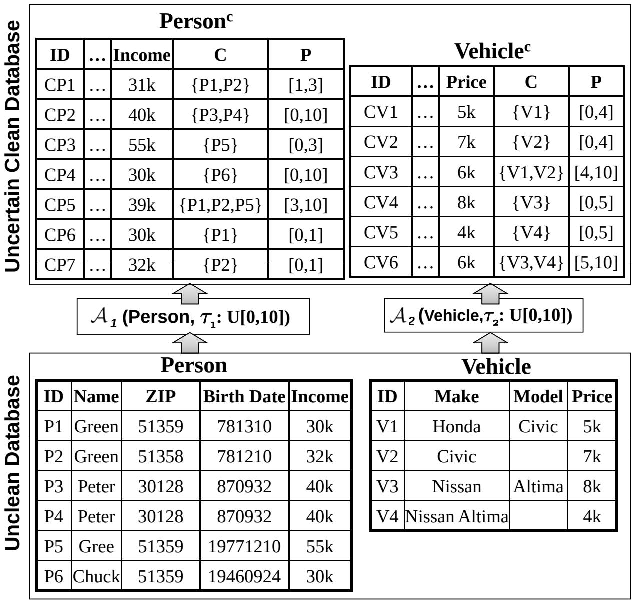
where $h(t, X) = 1$ if $\mathcal{A}(R, t) = X$, and 0 otherwise.

In the following, we define uncertain clean relation (*U-clean relation* for short) that encodes the possible repairs $\mathcal{X}$ of an unclean relation $R$ generated by a parameterized clustering algorithm $\mathcal{A}$.

DEFINITION 2. ***U-Clean Relation.*** *A U-clean relation, denoted $R^c$, is a set of c-records where each c-record is a representative record of a cluster of records in R. Attributes of $R^c$ are all attributes of Relation R, in addition to two special attributes: C and P. Attribute C of a c-record is the set of records identifiers in R that are clustered together to form this c-record. Attribute P of a c-record represents the parameter settings of the clustering algorithm $\mathcal{A}$ that lead to generating the cluster represented by this c-record.*

The parameter settings $P$ is represented as one or more intervals within the range of the algorithm parameter $\tau$. We interpret each $c$-record $r$ as a propositional variable, and each repair $X \in \mathcal{X}$ as a truth assignment for all $c$-records in $R^c$ such that $r =$ True if records in Attribute $C$ of $r$ form a cluster in $X$, and $r =$ False otherwise. Note that it is possible to have overlapping clusters represented by different $c$-records in $R^c$ since $R^c$ encapsulates more than one possible repair of $R$.

Figure 5 illustrates our model of possible repairs for two unclean relations Person and Vehicle. U-clean relations Person$^c$ and Vehicle$^c$ are created by clustering algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ using parameters $\tau_1$ and $\tau_2$, respectively. For brevity, we omit some attributes from Person$^c$ and Vehicle$^c$ (shown as dotted columns in Figure 5). Parameters $\tau_1$ and $\tau_2$ are defined on the real interval $[0, 10]$ with uniform distributions. We provide more details of the construction process in Section 3.2. Relations Person$^c$ and Vehicle$^c$ capture all repairs of the base relations corresponding to possible parameters values. For example, if $\tau_1 \in [1, 3]$, the resulting repair of Relation Person is equal to {{P1,P2}, {P3,P4}, {P5}, {P6}}, which is obtained using $c$-records in Person$^c$ whose parameter settings contain the interval $[1, 3]$. Moreover, the U-clean relations allow for identifying the parameter settings of the clustering algorithm that lead to generating a specific cluster of records. For example, the cluster {P1,P2,P5} is generated by algorithm $\mathcal{A}_1$ if the value of parameter $\tau_1$ belongs to the range $[3, 10]$.


Uncertain Clean Database
Unclean Database
$\mathcal{A}_1$ (Person, $\tau_1$: U[0,10])
$\mathcal{A}_2$ (Vehicle,$\tau_2$: U[0,10])

Person$^c$

| ID | … | Income | C | P |
|---|---|---|---|---|
| CP1 | … | 31k | {P1,P2} | [1,3] |
| CP2 | … | 40k | {P3,P4} | [0,10] |
| CP3 | … | 55k | {P5} | [0,3] |
| CP4 | … | 30k | {P6} | [0,10] |
| CP5 | … | 39k | {P1,P2,P5} | [3,10] |
| CP6 | … | 30k | {P1} | [0,1] |
| CP7 | … | 32k | {P2} | [0,1] |

Vehicle$^c$

| ID | … | Price | C | P |
|---|---|---|---|---|
| CV1 | … | 5k | {V1} | [0,4] |
| CV2 | … | 7k | {V2} | [0,4] |
| CV3 | … | 6k | {V1,V2} | [4,10] |
| CV4 | … | 8k | {V3} | [0,5] |
| CV5 | … | 4k | {V4} | [0,5] |
| CV6 | … | 6k | {V3,V4} | [5,10] |

Person

| ID | Name | ZIP | Birth Date | Income |
|---|---|---|---|---|
| P1 | Green | 51359 | 781310 | 30k |
| P2 | Green | 51358 | 781210 | 32k |
| P3 | Peter | 30128 | 870932 | 40k |
| P4 | Peter | 30128 | 870932 | 40k |
| P5 | Gree | 51359 | 19771210 | 55k |
| P6 | Chuck | 51359 | 19460924 | 30k |

Vehicle

| ID | Make | Model | Price |
|---|---|---|---|
| V1 | Honda | Civic | 5k |
| V2 | Civic | | 7k |
| V3 | Nissan | Altima | 8k |
| V4 | Nissan Altima | | 4k |

**Figure 5: Example Illustrating the Specialized Uncertainty Model**

## 3.2 Constructing U-clean Relations

Hierarchical clustering algorithms cluster records of an input relation $R$ in a hierarchy, which represents a series of possible clusterings starting from a clustering containing each record in a separate cluster, to a clustering containing all records in one cluster (e.g., Figure 6). The algorithms use specific criteria, usually based on a parameter of the algorithm, to determine which clustering to return.

Hierarchical clustering algorithms are widely used in duplicate detection. Examples include link-based algorithms (e.g., single-linkage, average-linkage and complete-linkage), hierarchical cut clustering [18], and CURE [20]. Other algorithms can be altered to allow producing hierarchical clustering of records such as the fuzzy duplicate detection framework introduced in [10], as we show later in this section. Hierarchical clustering is also used as a basis for other duplicate detection algorithms such as collective entity resolution [9], and deduplication under aggregate constraints [11].

Due to the nature of hierarchical clustering algorithms, only minor modifications are necessary to allow constructing U-clean relations as we discuss in the following case studies.

**Case Study 1: Link-based Hierarchical Clustering Algorithms**

Given an input unclean relation $R$ of $n$ records, a hierarchical linkage-based clustering algorithm is generally described based on two components: (1) a distance function $dist : 2^R \times 2^R \rightarrow \mathbb{R}$, where $2^R$ is the power set of $R$, that gives the distance between two disjoint clusters of records, and (2) a threshold $\tau$ such that any two

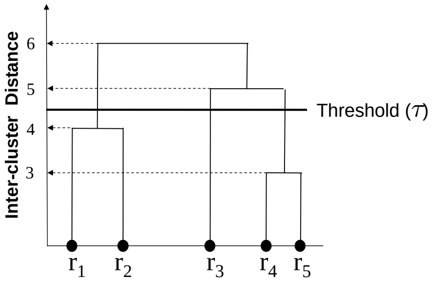


**Figure 6: Example of Link-based Hierarchical Clustering**

clusters $C_1$ and $C_2$ with $dist(C_1, C_2) > \tau$ are not linked (merged). Clusters are merged iteratively starting from all singleton clusters. At each iteration, the function $dist$ is used to pick the closest cluster pair to link. If the value of $dist$ of such pair is below the threshold $\tau$, the two clusters are merged by creating a parent cluster in the hierarchy composed of the union of the two original clusters. The distance between two records is determined through various functions such as Euclidian distance, Edit Distance, and Q-grams [16]. The distance between two clusters is an aggregate of the pair-wise distances. For example, in single-linkage [23], $dist$ returns the distance between the two closest records in the two clusters, while in complete-linkage, $dist$ is the distance between the two furthest records in the two clusters.

Figure 6 gives an example of the hierarchy generated by a linkage-based algorithm for the relation $R = \{r_1, \ldots, r_5\}$. The parameter $\tau$ represents a threshold on inter-cluster distances which are represented by the $Y$-axis. Different repairs are generated when applying the algorithm with different values of $\tau$. For example, at $\tau \in [0, 3]$, the produced repair is $\{\{r_1\}, \{r_2\}, \{r_3\}, \{r_4\}, \{r_5\}\}$, while at $\tau \in [3, 4]$, the produced repair is $\{\{r_1\}, \{r_2\}, \{r_3\}, \{r_4, r_5\}\}$.

We modify the link-based clustering algorithm to build U-clean relations as described in Algorithm 1. The algorithm performs clustering of records similar to the conventional greedy agglomerative clustering algorithm. However, we additionally create and store all $c$-records corresponding to the clusters linked at distances within the range $[\tau^l, \tau^u]$. We denote by $r[A]$ the value of Attribute $A$ of the record $r$. Initially, the algorithm creates a singleton cluster and its corresponding $c$-record for each record in $R$ (lines 1-5). The initial parameter settings of created $c$-records are the entire range $[\tau^l, \tau^u]$. The algorithm incrementally merges the closest clusters $C_i$ and $C_j$, and creates a $c$-record corresponding to the new cluster (lines 6-10). Additionally, we update the $c$-records corresponding to $C_i$ and $C_j$ as shown in lines 11-15. The algorithm terminates when the distance between the closest clusters exceeds $\tau^u$ or when all records are clustered together.

### Case Study 2:NN-Based Clustering

In [10], a duplicate detection algorithm based on nearest neighbor (NN) techniques is introduced. The algorithm introduced in [10] declares a set of records as duplicates whenever they represent a *compact set* that has *sparse neighborhood*. A set of records $S$ is compact if $\forall r \in S$, the distance between $r$ and any other record in $S$ is less than the distance between $r$ and any record not in $S$. The neighborhood growth of a record $r$, denoted $ng(r)$, is defined as the number of records with distance to $r$ smaller than double the distance between $r$ and its nearest neighbor. A set $S$ has a sparse neighborhood if its aggregated neighborhood growth $F_{r \in S}\ ng(r)$ is less than a threshold $\tau$, where $F$ is an aggregate function such as

**Algorithm 1** `U_Cluster`$(R(A_1, \ldots, A_m), \tau^l, \tau^u)$

```
Require: $R(A_1, \ldots, A_m)$: The unclean relation
Require: $\tau^l$: Minimum threshold value
Require: $\tau^u$: Maximum threshold value
1: Define a new singleton cluster $C_i$ for each record $r_i \in R$ (i.e., $C_i$ contains a unique identifier of $r_i$)
2: $\mathcal{C} \leftarrow \{C_1, \ldots, C_{|R|}\}$
3: for each $r_i \in R$ do
4:     Add $(r_i[A_1], \ldots, r_i[A_m], C_i, [\tau^l, \tau^u])$ to $R^c$
5: end for
6: while $(|\mathcal{C}| > 1$ and distance between the closest pair of clusters $(C_i, C_j)$ in $\mathcal{C}$ is less than $\tau^u)$ do
7:     $C_k \leftarrow C_i \cup C_j$
8:     Replace $C_i$ and $C_j$ in $\mathcal{C}$ with $C_k$
9:     $r_k \leftarrow$ get_representative_record$(C_k)$ {See Section 3.3}
10:    Add $(r_k[A_1], \ldots, r_k[A_m], C_k, [dist(C_i, C_j), \tau^u])$ to $R^c$
11:    if $(dist(C_i, C_j) < \tau^l)$ then
12:        Remove the $c$-records corresponding to $C_i$ and $C_j$ from $R^c$
13:    else
14:        Set the upper bounds of parameter settings of the $c$-records corresponding to $C_i$ and $C_j$ to $dist(C_i, C_j)$
15:    end if
16: end while
17: return $R^c$
```

$max$ or $average$.

Although the clustering algorithm in [10] is not presented as a hierarchical clustering algorithm, we prove in [8] that compact sets form a hierarchy (i.e, compact sets are necessarily nested). Such a fact allows using the NN-based clustering algorithm to provide a hierarchy of clusters as we describe in the following.

If the used aggregation function $F$ is $max$ (or any other monotone function w.r.t. to the size of a compact set), increasing $\tau$ results in a monotonic decrease of the number of clusters by merging two or more compact sets into one compact set. The reason is that for any two compact sets $S_i, S_j$ such that $S_i \subset S_j$, $\max_{r \in S_i}\ ng(r) \leq \max_{r \in S_j}\ ng(r)$. Thus, the NN-based clustering algorithm effectively constructs a hierarchy of compact sets where neighborhood spareness is used as the stopping condition.

In order to allow efficient construction of U-clean relations, we modify the NN-based clustering algorithm similar to link-based algorithms. We construct compact sets incrementally, starting with singleton compact sets until reaching compact sets of the maximum size allowed (using the same technique in [10]). Each compact set with aggregated neighborhood growth above $\tau^l$ and below $\tau^u$ is stored in $R^c$. For any two compact sets $S_i, S_j \in \mathcal{S}$ such that $S_i \subset S_j$ and they have the same neighborhood growth, we only store $S_j$ in $R^c$ in order to comply with a property of the algorithm in [10], which is to report the *largest* compact sets that satisfy the sparse neighborhood criterion. Parameter settings are maintained for each $c$-record similar to the linkage-based algorithms.

### Time and Space Complexity

In general, our uncertain hierarchical clustering algorithms have the same asymptotic complexity of the conventional algorithms. The reason is that we only add a constant amount of work to each iteration which is constructing $c$-records and updating their parameter settings (e.g., lines 9-15 in Algorithm 1).

Hierarchical clustering arranges records in the form of an $N$-ary tree. The leaf nodes in the tree are the records in the unclean relation $R$, while the internal nodes are clusters of records that contain two or more records. Let $n$ be the size of $R$, and $n'$ be the number of clusters containing two or more records (the number of internal nodes). The maximum value of $n'$ occurs when the tree is binary, in which $n'$ is equal to $n-1$. Thus, the total number of nodes in the clustering hierarchy is less than or equal to $n' + n = 2n - 1$. The size of Relation $R^c$ is equal to the number of the possible clusters which is bounded by $2n - 1$. It follows that the size of $R^c$ is linear in the number of records in $R$.

The number of repairs encoded by $R^c$ is equal to the number of internal nodes $n'$ as each internal node indicates merging multiple clusters together, which results in a new repair. Moreover, the maximum repair size is $n$ (the number of leaf nodes).

## 3.3 Representative Records of Clusters

Records in the same cluster within a repair indicate duplicate references to the same real-world entity. Queries that reason about attributes of entities usually require resolving potential conflicts between attributes of duplicate records. We assume that conflicts in attribute values of the cluster records are resolved deterministically using a user defined merging procedure (line 9 in Algorithm 1), which can be decided based on application requirements. For example, conflicting values of Attributes `Income` and `Price` in Figure 5 are resolved by using their average as a representative value.

Note that deterministically resolving conflicts in attribute values of records that belong to the same cluster may lead to loss of information and introduce errors in the generated repairs. In [5], authors tackled this problem by modeling uncertainty in merging records. The authors assume that a representative record for a cluster is a random variable whose possible outcomes are all members of the cluster. We see uncertainty in the merging operation as another level of uncertainty that could be combined in our framework. For the sake of clarity, we focus on uncertainty in clustering records. We describe how to extend our approach to handle uncertain merging in the full version of the paper [8].

# 4. QUERY PROCESSING

In this section, we describe how to support multiple query types under our model such as selection, projection, join and aggregation.

## 4.1 Semantics of Query Answering

We define relational queries over U-clean relations using the concept of *possible worlds semantics* [15]. More specifically, queries are semantically answered against individual clean instances of the dirty database that are encoded in input U-clean relations, and the resulting answers are weighted by the probabilities of their originating repairs. For example, consider a selection query that reports persons with `Income` greater than 35k considering all repairs encoded by Relation `Person`$^c$ in Figure 5. One qualified record is `CP3`. However, such record is valid only for repairs generated at the parameter settings $\tau_1 \in [0, 3]$. Therefore, the probability that record `CP3` belongs to the query result is equivalent to the probability that $\tau_1$ is within $[0, 3]$, which is $0.3$.

## 4.2 SPJ Queries

In this section, we define the selection, projection and join (SPJ) operators over U-clean relations.

Model closure under SPJ queries is important in order to allow query decomposition (i.e., applying operators to the output of other operators). To make our model closed under SPJ operations, we extend Attribute $C$ in U-clean relations to be a composition of multiple clusters, and extend Attribute $P$ to be a composition of multiple parameter settings of one or more clustering algorithms. Similar methods are proposed in [15], where each record is associated with a complex probabilistic event.

We interpret Attribute $C$ (and similarly Attribute $P$) of a base $c$-record $r$ as a propositional variable that is `True` for repairs containing $r$, and is `False` for all other repairs. We define Attribute $C$ (and similarly $P$) in U-clean relations resulting from SPJ queries as propositional formula in DNF over Attribute $C$ (similarly $P$) of the base U-clean relations. Note that the propositional formulae of Attributes $C$ and $P$ of a $c$-record $r$ are identical DNF formulae defined on the clusters and the parameter settings of the base $c$-records corresponding to $r$, respectively. We give examples in the following sections.

**SELECT** ID, Income
**FROM** Person$^c$
**WHERE** Income>35k

| ID | Income | C | P |
|---|---|---|---|
| CP2 | 40k | {P3,P4} | [0,10] |
| CP3 | 55k | {P5} | [0,3] |
| CP5 | 39k | {P1,P2,P5} | [3,10] |

(a)

**SELECT DISTINCT** Price
**FROM** Vehicle$^c$

| Price | C | P |
|---|---|---|
| 4k | {V4} | [0,5] |
| 5k | {V1} | [0,4] |
| 6k | {V1,V2}v{V3,V4} | [4,10]v[5,10] |
| 7k | {V2} | [0,4] |
| 8k | {V3} | [0,5] |

(b)

**SELECT** Income, Price
**FROM** Person$^c$ , Vehicle$^c$
**WHERE** Income/10 >= Price

| Income | Price | C | P |
|---|---|---|---|
| 40k | 4k | {P3,P4} ^ {V4} | $\tau_1$:[0, 10] ^ $\tau_2$:[0,5] |
| 55k | 5k | {P5} ^{V1} | $\tau_1$:[0, 3] ^ $\tau_2$:[0,4] |
| 55k | 4k | {P5} ^ {V4} | $\tau_1$:[0, 3] ^ $\tau_2$:[0,5] |

(c)

**Figure 7: Relational Queries (a) Selection (b) Projection (c) Join**

SPJ operators that are applied to U-clean relations are *conceptually* processed against all clean instances represented by the input U-clean relations, and the resulting instances are re-encoded into an output U-clean relation. We add a superscript $u$ to the operators symbols to emphasize awareness of the *uncertainty* encoded in the U-clean relations. In the following, we show how to efficiently evaluate SPJ queries without an exhaustive processing of individual repairs, based on the concept of *intensional query evaluation* [15].

### *4.2.1 Selection*

We define the selection operator over U-clean relations, denoted $\sigma^u$, as follows: $\sigma^u_p(R^c) = \{r : r \in R^c \wedge p(r) = True\}$, where $p$ is the selection predicate. That is, a selection query $\sigma^u_p(R^c)$ results in a U-clean relation containing the $c$-records in $R^c$ that satisfy the predicate $p$. The operator $\sigma^u$ does not change Attributes $C$ or $P$ of the resulting $c$-records. The resulting U-clean relation encodes all the clean instances represented by $R^c$ after filtering out all records not satisfying $p$.

For example, Figure 7(a) shows the result of a selection query posed against Relation `Person`$^c$ in Figure 5, where we are interested in finding persons with income greater than 35k. The query produces three $c$-records that are identical to the input $c$-records `CP2`, `CP3`, and `CP5`.

### *4.2.2 Projection*

We define the projection operator $\Pi^u$ over a U-clean relation as follows. The expression $\Pi^u_{A_1,\ldots,A_k}(R^c)$ returns a U-clean relation that encodes projections of all clean instances represented by $R^c$ on Attributes $A_1, \ldots, A_k$. Therefore, the schema of the resulting U-clean relation is $(A_1, \ldots, A_k, C, P)$. Under bag semantics, duplicate $c$-records are retained. Hence, Attributes $C$ and $P$ of the projected $c$-records remain unchanged. Under set semantics, $c$-records with identical values with respect to Attributes $A_1, \ldots, A_k$ are reduced to only one $c$-record with Attributes $C$ and $P$ com-

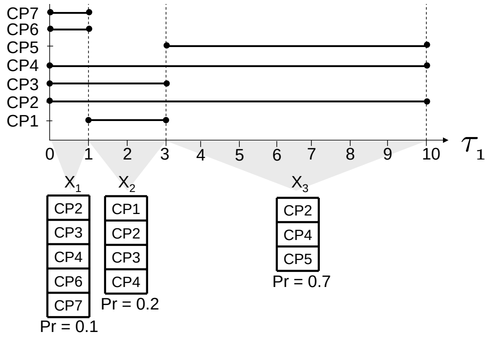


**Figure 8: Distribution of Possible Repairs in Relation `Person`$^c$**

puted as follows. Let $\acute{r} \in \Pi^u_{A_1,\ldots,A_k}(R^c)$, where $\acute{r}$ is a projected $c$-record corresponding to duplicate $c$-records $\{r_1, \ldots, r_m\} \subseteq R^c$. Attribute $C$ of $\acute{r}$ is equal to $\bigvee_{i=1}^m r_i[C]$ and Attribute $P$ of $\acute{r}$ is equal to $\bigvee_{i=1}^m r_i[P]$.

For example, Figure 7(b) shows the results of a projection query (under set semantics) posed against Relation `Vehicle`$^c$ in Figure 5, where we are interested in finding the distinct car prices. The only duplicate $c$-records w.r.t. Attribute `Price` are `CV3` and `CV6`.

### *4.2.3 Join*

We define the join operator $\bowtie^u$ over two U-clean relations as follows. The expression $(R_i^c \bowtie_p^u R_j^c)$ results in a U-clean relation that contains all pairs of $c$-records in $R_i^c$ and $R_j^c$ that satisfy the join predicate $p$. Additionally, we compute Attributes $C$ and $P$ of the resulting $c$-records as follows. Let $r_{ij}$ be the result of joining $r_i \in R_i^c$ and $r_j \in R_j^c$. Attribute $C$ of $r_{ij}$ is equal to the conjunction of values of Attribute $C$ of both $r_i$ and $r_j$. That is, $r_{ij}[C] = r_i[C] \wedge r_j[C]$. Similarly, $r_{ij}[P] = r_i[P] \wedge r_j[P]$. Therefore, the schema of the resulting U-clean relation contains only one occurrence of Attributes $C$ and $P$. The resulting U-clean relation encodes the results of joining each clean instance stored in $R_i^c$ with each clean instance stored in $R_j^c$.

For example, Figure 7(c) shows the results of a join query on Relations `Person`$^c$ and `Vehicle`$^c$ in Figure 5. The query finds which car is likely to be purchased by each person by joining a person with a car if 10% of the person's income is greater than or equal to the car's price. Note that the parameter settings of $c$-records in the join results involve two parameters: $\tau_1$ and $\tau_2$. Therefore, we precede each interval in Attribute $P$ with the referred parameter to avoid ambiguous settings.

## 4.3 Aggregation Queries

The aggregation query $Agg(R^c, expr)$ uses the function $Agg$ (such as $sum$, $count$, and $min$) to aggregate the value of the expression $expr$ over all $c$-records in $R^c$. Examples of $expr$ include a single attribute in $R^c$, or a function defined on one or more attributes. The result of an aggregation query against one clean database instance is a single scaler value. However, in our settings, Relation $R^c$ encodes multiple possible clean instances. Hence, the answer of an aggregation query over $R^c$ is a probability distribution over possible answers, each of which is obtained from one or more clean possible instances.

To simplify the discussion, we assume that the aggregate query involves a base U-clean relation $R^c$ that is generated by a clustering algorithm $\mathcal{A}$ using a single parameter $\tau$. We discuss in the extended version of the paper [8] how to answer aggregation queries over U-clean relations resulting from SPJ queries.

For example, consider the aggregation query $average$(`Person`$^c$, `Income`), where we are interested in finding the average of persons' incomes, given the possible repairs represented by `Person`$^c$ in Figure 5. Figure 8 shows the possible repairs of Relation `Person`, which are $\{CP2, CP3, CP4, CP6, CP7\}$, $\{CP1, CP2, CP3, CP4\}$ and $\{CP2, CP4, CP5\}$ whose probabilities are 0.1, 0.2 and 0.7, respectively. The aggregate value for the three repairs are 37.4k, 39k and 36.33k, respectively. Hence, the query answer is the following discrete probability distribution: $\Pr(average = 37.4k) = 0.1$, $\Pr(average = 39k) = 0.2$, $\Pr(average = 36.33k) = 0.7$.

A straightforward algorithm to answer aggregation queries over a U-clean relation $R^c$ is described as follows.

1. Identify the distinct end points $t_1, \ldots, t_m$ (in ascending order) that appear in Attribute $P$ of all $c$-records in $R^c$. Define $V_i$ to be the interval $[t_i, t_{i+1}]$ for $1 \leq i \leq m-1$.
2. For each interval $V_i$ :
   (a) Obtain the corresponding repair $X_i = \{r : r \in R^c \wedge V_i \subseteq r[P]\}$.
   (b) Evaluate Function $Agg$ over $X_i$.
   (c) Compute the probability of $X_i$: $\int_{V_i} f_\tau(x)dx$.
3. Compute the probability of each value of $Agg$ as the sum of probabilities of the repairs corresponding to such a value.

For example, for the aggregation query $average$(`Person`$^c$, `Income`), we extract the end points in Attribute $P$ of `Person`$^c$, which are $\{0, 1, 3, 10\}$ as shown in Figure 8). The corresponding intervals $[0,1]$, $[1,3]$, and $[3,10]$ represent the repairs $X_1$, $X_2$ and $X_3$, respectively. We compute the aggregate value corresponding to each repair by evaluating Function $Agg$ over the $c$-records in this repair. Finally, we report each aggregate value along with the sum of probabilities of its corresponding repairs. The complexity of the described algorithm is $O(n^2)$ due to evaluating the function $Agg$ over individual repairs (recall that the number of repairs is $O(n)$ and the size of a repair is $O(n)$ as shown in Section 3.2). In the remainder of this section, we show how to reduce such complexity to $O(n \log n)$.

We employ a method to incrementally evaluate the aggregate function $Agg$, which is based on the concept of *partial aggregate states* [26, 3]. Such a method is based on defining an aggregate state for a given subset of the items to be aggregated. States of disjoint subsets are aggregated to obtain the final result. More specifically, three functions have to be defined [26]: Function $init\_state$ that initializes a state corresponding to a specific set of items, Function $merge\_states$ that merges two partial states into one state, and Function $finalize\_state$ that obtains the final answer corresponding to a state. For example, in the case of the aggregate function $average$, a partial state consists of a pair $(sum, count)$. The initialization of the empty set returns the state $(0, 0)$, while initialization of a set with a single item $v$ returns the state $(v, 1)$. The merging of two states $(sum1, count1)$ and $(sum2, count2)$ returns the state $(sum1 + sum2, count1 + count2)$. The finalization function returns the value of $sum/count$.

We define a B-tree index, denoted $I$, over the parameter space $[\tau^l, \tau^u]$ such that each interval $V_i$ is represented as a leaf node in $I$ (denoted as $V_i$ as well). Additionally, we associate a partial aggregate state to each node $l$ in $I$, denoted $l.state$. We construct $I$ such that the final aggregate value at each interval $V_i$ is computed by merging the state $V_i.state$ with states of all ancestors of $V_i$.

**Algorithm 2** Aggregate ($R^c, expr, init\_state, merge\_states, finalize\_state$)

```
Require: $R^c$: An input U-clean relation
Require: $expr$: An expression over attributes of $R^c$
Require: $init\_state$,$merge\_states$, $finalize\_state$: Functions for manipulating partial aggregate states
 1: Define an index $I$ over the space of the clustering algorithm parameter
 2: Initialize $I$ to have one node ($I.root$) covering the entire parameter space
 3: $I.root.state \leftarrow init\_state(\phi)$
 4: Define a set $D$ (initially empty)
 5: Define a partial state $record\_state$
 6: for each $r \in R^c$ do
 7:     $record\_state \leftarrow init\_state(\{expr(r)\})$
 8:     Update_Index($I.root$, $r[P]$, $record\_state$, $merge\_states$ )
 9: end for
10: for each node $l \in I$, using pre-order traversal do
11:     if $l \neq I.root$ then
12:         $l.state \leftarrow merge\_states(l.state, l.parent.state)$
13:     end if
14:     if $l$ is a leaf node then
15:         $Agg\_value \leftarrow finalize\_state(l.state)$
16:         $Prob \leftarrow \Pr(l)$ {see discussion in Section 5.1}
17:         Add $(Agg\_value, Prob)$ to $D$
18:     end if
19: end for
20: Merge pairs in $D$ with the same $Agg\_value$ and sum up their $Prob$
21: return $D$
```

**Algorithm 3** Update_Index ($l, P, record\_state, merge\_states$)

```
Require: $l$: a node in an index
Require: $P$: parameter interval to be updated
Require: $record\_state$: a new state to be merged within the interval $P$
Require: $merge\_states$: A function to merge multiple states
 1: if range of node $l$ is entirely contained in $P$ then
 2:     $l.state \leftarrow merge\_states(l.state, record\_state)$
 3: else if $l$ is an internal node and $l$ intersects with $P$ then
 4:     for each child node $l'$ of $l$ do
 5:         Update_Index ($l', P, record\_state, merge\_states$)
 6:     end for
 7: else if $l$ is an leaf node and range of $l$ intersects $P$ then
 8:     Split $l$ into multiple nodes such that only one new leaf node $l'$ is contained in $P$ and the other node(s) are disjoint from $P$
 9:     Set states of all new leaf nodes to the state of the old leaf node $l$
10:     $l'.state \leftarrow merge\_states(l'.state, record\_state)$
11: end if
```

Algorithms 2 and 3 outline our procedure to obtain the PDF of the aggregate value. Initially, the entire parameter space is covered by one node in the index, named $I.root$. The state of $I.root$ is initialized to the state of the empty set (e.g., $(0, 0)$ in case of the function $average$). For each $c$-record in $R^c$, the procedure Update_Index is invoked. Update_Index recursively traverses the index $I$ starting from the root node. For each node $l$, if the associated parameter range is completely covered by the interval $P$, we update the state of $l$, otherwise, if $l$ is an internal node, we recursively process its children nodes. If $l$ is a leaf node, we split it into multiple nodes such that one of the new nodes is contained in the interval $P$ (and thus we update its state accordingly), and the other node(s) are disjoint from $P$ (and thus their states are not changed). Whenever a node is split (as it becomes full, or due to the condition at line 7 in Algorithm 3), new nodes will have the state of the original node. If a new root is introduced, its state is set to $init\_state(\phi)$.

Once all $c$-records are consumed (after line 9 in Algorithm 2), we traverse the index $I$ in pre-order, and we merge the state of each node with the state of its parent. For each leaf node, a pair consisting of the aggregate value and its corresponding probability is stored. Finally, pairs are grouped by the aggregate value, and the probabilities of the same values are summed up. The resulting pairs represent the PDF of the aggregate function. We prove in the extended version of the paper [8] that our algorithm has a complexity of $O(n \log n)$ , where $n$ is the number of $c$-records in $R^c$.

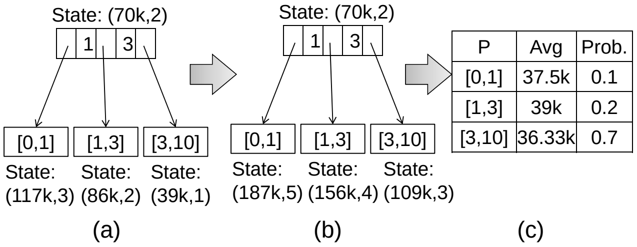


**Figure 9: Example Aggregation Query (a) Index $I$ after Line 9 in Alg. 2 (b) Index $I$ after Line 19 in Alg. 2 (c) PDF of the Aggregate Value**

Figure 9 shows an example to illustrate this procedure for the aggregation query $average$(Person$^c$, Income). We start with a node covering the parameter range $[0, 10]$, which is initialized to state $(0, 0)$. After reading the first $c$-record CP1, we split the node $[0, 10]$ into three leaf nodes $[0, 1]$, $[1, 3]$, and $[3, 10]$ associated with the states $(0, 0)$, $(31k, 1)$ and $(0, 0)$, respectively. At the same time, a new root is introduced with the state $(0, 0)$. When reading the next $c$-record CP2, we update the state of the root node to $(40k, 1)$. We repeat the same process when reading the remaining $c$-records. The final B-tree is shown in Figure 9(a). Then, we merge the state of each node with the states of its ancestors (Figure 9(b) ). The final PDF is then derived from the states of leaf nodes (Figure 9(c) ).

# 5. IMPLEMENTATION IN RDBMS

In this section, we show how to implement U-clean relations and query processing inside relational database systems. We also provide new queries that reason about uncertainty of repairing.

## 5.1 Implementing U-clean Relations

We implement Attributes $C$ and $P$ in a relational database as abstract data types (ADTs). Attribute $C$ is encoded as a set of (ORed) clauses, each of which is a set of (ANDed) clusters. Attribute $P$ of a U-clean relation $R^c$ is encoded as an array of hyper-rectangles in the $d$-dimensional space, where $d$ is the number of parameters of the used clustering algorithms. Each hyper-rectangle is represented as $d$ one-dimensional intervals.

Executing SPJ queries requires manipulation of Attributes $C$ and $P$ according to the discussion in Section 4.2. The selection and projection (under bag semantics) operators do not alter the values of $C$ and $P$, and hence no modifications are necessary to these operators in relational DBMSs. On the other hand, the join operator modifies Attributes $C$ and $P$ to be the conjunctions of the joined $c$-records attributes. $C$ and $P$ of the results are computed through functions $ConjC(C_1, C_2)$ and $ConjP(P_1, P_2)$, where $C_1$ and $C_2$ are record clusters, and $P_1$ and $P_2$ are parameter settings of clustering algorithm(s). We implement the functions $ConjC$ and $ConjP$ such that they return the conjunction of their inputs in DNF.

In the current implementation, we do not provide native support to projection with duplicate elimination (i.e., using the Distinct keyword). However, we realize projection with duplicate elimination through group-by queries. We implement two functions $DisjC(C_1, \ldots, C_n)$ and $DisjP(P_1, \ldots, P_n)$ to obtain the disjunction of clusters $C_1, \ldots, C_n$ and parameter settings $P_1, \ldots, P_n$, respectively. Performing projection with duplicate elimination of a U-clean relation `UR` on a set of attributes `A1,...,Ak` is equivalent to the following SQL query:

```
SELECT A1,...,Ak, DisjC(C), DisjP(P)
FROM UR
GROUP BY A1,...,Ak
```

This query effectively projects Relation `UR` on Attributes `A1,...,Ak` and computes the disjunctions of Attributes $C$ and $P$ of the duplicate $c$-records.

In the following, we give a list of operations that reason about the possible repairs encoded by a U-clean relation to allow new probabilistic query types that are described in Section 5.2.

- $Contains(P, x)$ returns `True` iff the parameter settings $P$ contain a given parameter setting $x$.
- $ContainsBaseRecords(C, S)$ returns `True` iff a set of base records identifiers $S$ is contained in a cluster $C$.
- $Prob(P, f_{\tau_1}, \dots, f_{\tau_d})$ computes the probability that a $c$-record with parameter settings $P$ belong to a random repair. $f_{\tau_1}, \dots, f_{\tau_d}$ are the probability distribution functions of the clustering algorithms parameters $\tau_1, \dots, \tau_d$ that appear in $P$.
- $MostProbParam(\texttt{UR}, f_{\tau_1}, \dots, f_{\tau_d})$ computes the parameter settings corresponding to the most probable repair of a U-clean relation `UR`, given the parameters PDFs $f_{\tau_1}, \dots, f_{\tau_d}$.

We describe how to efficiently implement the functions $Prob$ and $MostProbParam$ as follows.

### 5.1.1 Implementing Function $Prob$

$Prob$ determines the membership probability of a $c$-record, given its parameter settings $P$, denoted as $\Pr(P)$. For a base U-clean relation that involve a single clustering algorithm parameter $\tau$ with PDF $f_\tau$, this probability is equal to $\int_P f_\tau(x)dx$. For U-clean relations resulting from SPJ queries, Attribute $P$ involves $d$ parameters and is represented as a union of $h$ hyper-rectangles, denoted $H_1, \dots, H_h$, each of which is $d$ dimensional. We first divide the parameters space into a number of disjoint hyper-rectangles (called *cells*) as follows. For each parameter $\tau_i, 1 \leq i \leq d$, we extract the end points of each hyper-rectangle w.r.t. to $\tau_i$. The resulting points divide the space of $\tau_i$ into at most $2h+1$ intervals, and thus the space of all parameters is partitioned into at most $(2h+1)^d$ disjoint cells, denoted $\{L_1, L_2, \dots\}$. The probability of a cell $L_j$ is defined as follows.

$$\Pr(L_j) = \prod_{i=1}^{d} \int_{L_j.\tau_i^l}^{L_j.\tau_i^u} f_{\tau_i}(x)dx \qquad (2)$$

where $L_j.\tau_i^l$ and $L_j.\tau_i^u$ indicate the lower and upper values of parameter $\tau_i$ in cell $L_j$, respectively. Clearly, for each pair $(H_l, L_j)$, $L_j$ can only be either contained in $H_l$ or disjoint from $H_l$. Additionally, each $H_l$ is completely covered by one or more cells. Thus, we compute $\Pr(P)$ as follows.

$$\Pr(P) = \sum_j con(L_j, P) \Pr(L_j) \qquad (3)$$

where $con(L_j, P)$ is an indicator function that returns 1 if $L_j$ is contained in any hyper-rectangle of $P$, and 0 otherwise.

### 5.1.2 Implementing Function $MostProbParam$

For base U-clean relations, determining the most probable repair can be done efficiently by scanning $c$-records in $R^c$, and extracting all end points of their parameter settings. Distinct end points split the parameter space $[\tau^l, \tau^u]$ into multiple intervals $V_1, \dots, V_m$ corresponding to the possible repairs. For example, in Figure 8, the possible repairs are $X_1$, $X_2$, and $X_3$ corresponding to the intervals $[0, 1]$, $[1, 3]$, and $[3, 10]$, respectively. The probability of each repair is computed based on its corresponding parameter settings. Function $MostProbParam$ returns the interval of the repair with the highest probability (e.g., $[3, 10]$ in Figure 8). The overall complexity of the function $MostProbParam$ is $O(n \log n)$ (mainly, due to sorting the end points of parameter settings).

In U-clean relations resulting from SPJ queries, Attributes $C$ and $P$ are represented as propositional formulae in DNF. Therefore, our algorithm needs to be modified to consider such a representation, as we show in the extended version of the paper [8].

## 5.2 Other Query Types

In this section, we describe multiple meta-queries that are defined over our uncertainty model. Specifically, the queries we describe in this section explicitly use Attributes $P$ and $C$ to reason about the possible repairs modeled by U-clean relations.

### 5.2.1 Extracting Possible Clean Instances

Any clean instance encoded in a U-clean relation can be constructed efficiently given the required parameters values of the clustering algorithm(s). For a base U-clean relation $R^c$, the clean instance at parameter value $t$ is equal to $\{r[A_1, \dots, A_m] : r[A_1, \dots, A_m, P, C] \in R^c \wedge t \in P\}$.

Extracting the clean instance corresponding to a given parameter value $x$ can be performed through a selection query with the predicate $Contains(P, x)$. For example, assume that we need to extract the clean instance of the output U-clean relation in Figure 7(b) corresponding to the parameter setting $\tau_2 = 4.1$. This clean instance is computed using the following SQL query:

```
SELECT Price
FROM ( SELECT Price , DisjP(C) AS C, DisjP(P) AS P
       FROM Vehicle^c
       GROUP BY Price)
WHERE Contains(P, 4.1)
```

which results in the tuples $4k$, $6k$, and $8k$.

It is possible to speed up extraction of clean instances by indexing the $c$-records based on their parameter settings. More specifically, we create a $d$-dimensional R-tree index over the space of possible settings of parameters $\tau_1, \dots, \tau_d$. The parameter settings of a $c$-record $r$ is generally a union of $d$-dimensional hyper-rectangles. For each $c$-record $r \in R^c$, we insert its hyper-rectangles into the R-tree, and label them with an identifier of $r$. To extract the repair at $\tau_1 = t_1, \dots, \tau_d = t_d$, we search the R-tree for hyper-rectangles that contain the point $(t_1, \dots, t_d)$ and report the associated $c$-records.

### 5.2.2 Obtaining the Most Probable Clean Instance

An intuitive query is to extract the clean instance with the highest probability, as encoded in a given U-clean relation. It is possible to answer this query using two functions, namely $Contains$ and $MostProbParam$, through a selection SQL query. For example, assume that a user requests the most probable repair from Relation `Person`$^c$ which is shown in Figure 5. This query can be answered using the following SQL query:

SELECT ID, Name, ZIP, Income, BirthDate
FROM $\texttt{Person}^c$
WHERE $Contains(P, MostProbParam(\texttt{Person}^c, U(0, 10)))$

Note that $MostProbParam$ is evaluated only once during the entire query and thus the cost incurred by this function is paid only once.

### 5.2.3 Finding $\alpha$-certain $c$-records

We consider a query that finds $c$-records that exhibit a degree of membership certainty above a given threshold $\alpha$. We call this type of queries an $\alpha$-certain query. This query type can be answered by issuing a selection query with the predicate $Prob(P, f_{\tau_1}, \ldots, f_{\tau_d}) \geq \alpha$. For example, consider a 0.5-certain query over the relation in Figure 7(c). This query is answered using the following SQL query:

SELECT Income, Price, $ConjC(\texttt{PC}.C, \texttt{VC}.C), ConjP(\texttt{PC}.P, \texttt{VC}.P)$
FROM $\texttt{Person}^c$ PC , $\texttt{Vehicle}^c$ VC
WHERE Income/10 >= Price
AND $Prob(ConjP(\texttt{PC}.P, \texttt{VC}.P), U(0, 10), U(0, 10)) >= 0.5$

This SQL query reports only the first $c$-record in Figure 7(c), which has a membership probability of 0.5.

Note that $\alpha$-certain queries can be considered a generalization of consistent answers in inconsistent database [6]. That is, setting $\alpha$ to 1 retrieves only $c$-records that have absolute certain membership. The resulting $c$-records represent *consistent* answers in the sense that they exist in all possible repairs.

### 5.2.4 Probability of Clustering Records

We show how to compute the probability that multiple records in an unclean relation $R$ belong to the same cluster (i.e., declared as duplicates). For example, consider a query requesting the probability that two records P1 and P2 from Relation Person are clustered together according to the repairs encoded in U-clean relation $\texttt{Person}^c$ (Figure 5). The probability of clustering a set of records is equal to the sum of probabilities of repairs in which this set of records is clustered together. To compute this probability, we first select all $c$-records whose Attribute $C$ contains all query records (e.g., P1 an P2). Values of Attribute $C$ of the selected $c$-records are overlapping since they all contain the query records. Consequently, the selected $c$-records are exclusive (i.e., cannot appear in the same repair) and the clustering probability can be obtained by summing probabilities of the selected $c$-records:

$$\Pr(\text{clustering } r_1, \ldots, r_k) = \sum_{r \in R^c : \{r_1, \ldots, r_k\} \subseteq r[C]} \Pr(r[P]) \quad (4)$$

For example, the probability of clustering records P1 and P2 is obtained using the following query:

SELECT $Sum(Prob(P, U(0, 10)))$
FROM $\texttt{Person}^c$
WHERE $ContainsBaseRecords(C,$ 'P1,P2')

which returns the probability 0.9.

It is worth mentioning that the way we obtain clustering probabilities is substantially different from other approaches that computes the matching probabilities of records pairs. For example, in [17], Fellegi and Sunter derive the probability that two records are duplicates (i.e., match each other) based on the similarity between their attributes. Unlike our approach, probabilities of record pairs are computed in [17] in isolation of other pairs, which may lead to inconsistencies. For example, the pair $r_1, r_2$ may have a matching probability of 0.9, and the pair $r_2, r_3$ has a matching probability of 0.8, while the matching probability of the pair $r_1, r_3$ is equal to 0. Our approach avoids such inconsistencies by deriving pair-wise clustering probabilities based on the uncertain output of a clustering algorithm, which by definition resolves such inconsistencies. Moreover, our approach can obtain the matching probability of more than two records.

## 6. EXPERIMENTAL EVALUATION

In our experiments, we focus on the scalability of our approach compared to the deterministic deduplication techniques. We show in this section that although our proposed probabilistic cleaning framework generalizes current deterministic techniques and allows for richer probabilistic queries, it is still scalable and the time and space overheads are not significant, which warrants adopting our approach in realistic settings. We also show that queries over U-clean relations can be answered efficiently using our algorithms.

### 6.1 Setup

All experiments were conducted on a SunFire X4100 server with Dual Core 2.2GHz processor, and 8GB of RAM. We implemented all functions in Section 5.1 as user defined functions (UDFs) in PostgreSQL DBMS [3]. We used the synthetic data generator that is provided in the Febrl project [13], which produces one relation, named Person, that contains persons data (e.g., given_name,surname, address, phone, age). Datasets generated using Febrl exhibit the content and statistical properties of real-world datasets [12], including distributions of the attributes values, error types, and error positions within attribute values. Our experiments parameters are as follows:

- the number of records in the input unclean relation (default is 100,000),
- the percentage of duplicate records in the input relation (default is 10%), and
- the width of the parameter range used in the duplicate detection algorithms (default is 2, which is 10% of width of the broadest possible range according to the distribution of the pair-wise distance values). We assume that the parameters have uniform distributions. For deterministic duplicate detection, we always use the expected parameter value.

Our implementation of deterministic and uncertain duplicate detection is based on the single-linkage clustering (S.L.) [23], and the NN-based clustering algorithm using the function $max$ for aggregating neighborhood growths [10]. Deduplication algorithms are executed in memory. All queries, except aggregation queries, are executed through SQL statements submitted to PostgreSQL. Aggregation queries are processed by an external procedure that implements the algorithms described in Section 4.3. All queries are performed over a single U-clean relation, named $\texttt{Person}^c$, which is generated by uncertain deduplication of Person. Each query is executed five times and the average running time is recorded. We report the following metrics in our experiments:

- The running time of deterministic and uncertain clustering algorithms. The reported times do not include building the similarity graph, which is performed by Febrl [13].
- The sizes of the produced relations.

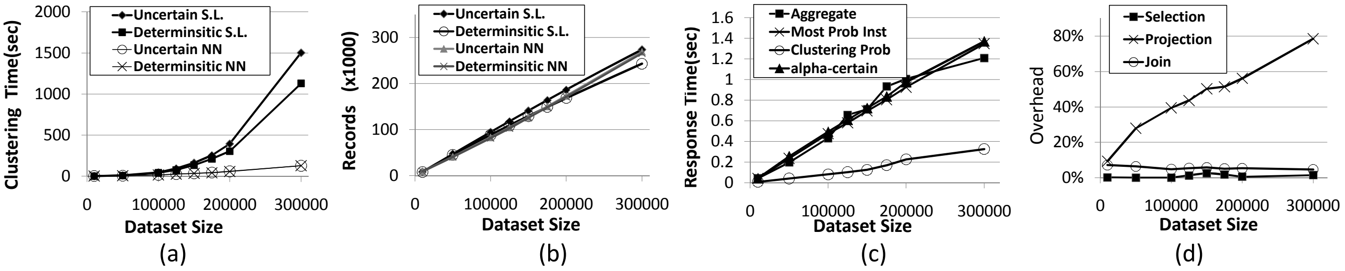


**Figure 10:** **Effect of Dataset Size on (a) Clustering Time (b) Output Size (c) Queries Running Times (d) SPJ Queries Overhead**

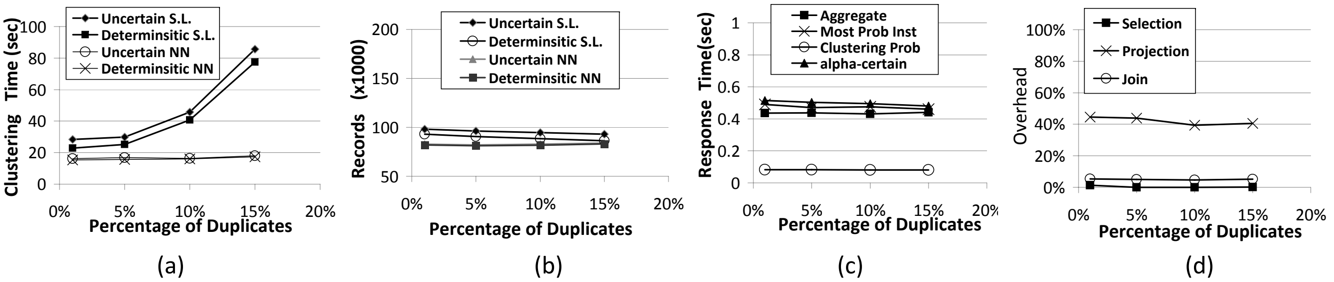


**Figure 11:** **Effect of Duplicates Percentage on (a) Clustering Time (b) Output Size (c) Queries Running Times (d) SPJ Queries Overhead**

- The response times of an aggregate query using the *count* function, and the probabilistic queries in Section 5.2 against Relation $\texttt{Person}^c$ constructed by the S.L. algorithm. The threshold $\alpha$ is set to 0.5 in $\alpha$-certain queries. In clustering probability queries, we use two random records as query arguments. We omit queries for extracting clean instances as they have almost identical response times to obtaining the most probable clean instance.

- The relative overhead of maintaining Attributes $C$ and $P$ in U-clean relations during selection, projection, and join queries as defined in Section 4.2 (we call such queries uncertain queries). We compare the uncertain queries to regular SPJ queries that do not use, compute, or return Attributes $C$ and $P$ (we call them base queries). The base selection query returns Attribute `Age` of all records, while the uncertain selection query returns Attributes `Age`, $C$ and $P$. The base join query performs a self-join over $\texttt{Person}^c$ to join $c$-records with the same `Surname` and different `ID`. The uncertain join query additionally computes Attributes $C$ and $P$ of the results. The base projection query (with duplicate elimination) projects Relation $\texttt{Person}^c$ on Attribute `surname` while ignoring Attributes $C$ and $P$. On the other hand, the uncertain projection computes Attributes $C$ and $P$ of the results as described in Section 5.1.

## 6.2 Results

We observe that the overhead in execution time of the uncertain deduplication, compared to the deterministic deduplication, is less than 30% in case of the S.L. algorithm, and less than 5% in case of the NN-based clustering algorithm. The average overhead in space requirements is equal to 8.35%, while the maximum overhead is equal to 33%. We also note that extracting a clean instance takes less than 1.5 seconds in all cases which indicates that our approach is more efficient than restarting the deduplication algorithm whenever a new parameter setting is requested.

The effect of changing the experiments parameters are shown in Figures 10, 11 and 12. We discuss these effects as follows:

**Effect of Dataset Size** (Figure 10): The overhead of the uncertain S.L. algorithm is almost fixed at 20% compared to the deterministic version. The running times of both versions of the NN-based algorithm are almost identical. Output sizes (Figure 10(b) ) and responses times of queries (Figure 10(c) ) exhibit linear (or near-linear) increase with respect to dataset size.

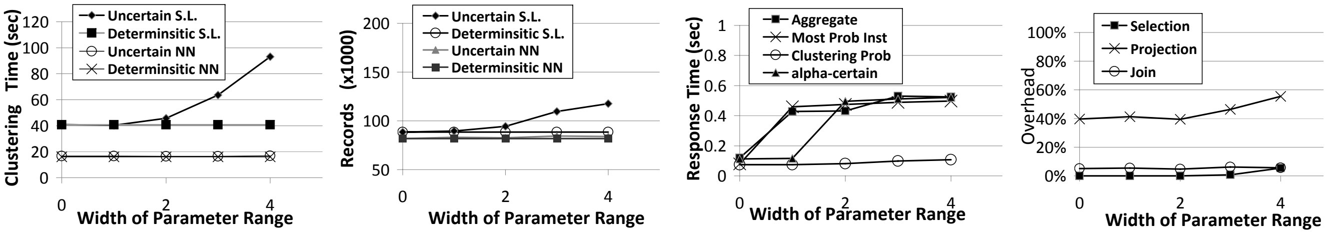
The overhead of processing SPJ queries varies among query types (Figure 10(d) ). Selection queries suffer from the lowest overhead (almost zero) because the only extra operation is converting data types of Attributes $C$ and $P$ into string format in output. Join queries have almost fixed relative overhead (about 5% in all cases) due to the constant time consumed in computing Attributes $C$ and $P$ per record in the join results. The projection query suffers from an overhead that increases linearly with the relation size due to evaluating the aggregate functions $DisjC$ and $DisjP$.

**Effect of Percentage of Duplicates** (Figure 11): The overhead of executing uncertain S.L. algorithm remains low (30% at most) as the percentage of duplicates rises (Figure 11(a) ). The uncertain NN-based algorithm has almost no overhead regardless of the amount of duplicates. The output size slightly declines at higher percentages of duplicates due to the increasing number of merged records (Figure 11(b) ). Produced clusters mainly consist of singletons. Hence, query response times are hardly affected by the increased percentage of duplicates (Figures 11(c) and 11(d) ).

**Effect of Width of Parameter Range** (Figure 12): The running times and the output sizes of the deterministic clustering algorithms do not change because the parameter value remains fixed at the expected value. In contrast, the running times of the uncertain clustering algorithms increase due to having greater upper bounds of parameters, which results in testing and clustering additional records. The output size also grows as more candidate clusters are emitted to the output U-clean relation (Figure 12(b) ). Consequently, queries suffer from increased response times (Figures 12(c) and 12(d) ).

# 7. RELATED WORK

A number of integrated data cleaning systems have been proposed with different focuses and goals. For example, AJAX [19]



(a) (b) (c) (d)

**Figure 12: Effect of Parameter Range on (a) Clustering Time (b) Output Size (c) Queries Running Times (d) SPJ Queries Overhead**

is an extensible framework attempting to separate the logical and physical levels of data cleaning. The logical level supports the design of the data cleaning workflow and specification of cleansing operations performed, while the physical level regards their implementation. IntelliClean [25] is a rule based approach to data cleaning that focuses on duplicates elimination. Such approaches do not capture uncertainty in the deduplication process.

The ConQuer system [5] addresses the deduplication problem in existence of uncertainty in the merging of cluster members. Each entity (i.e., cluster) is assumed to be equal to one of its member tuples. However, unlike our approach, the authors assume that clustering of records is deterministically performed prior to applying the uncertain merging.

A related problem is discussed in [29], which is to integrate two lists of items that possibly contains duplicate references to the same real world entities. The authors presents an XML-based uncertainty model to capture multiple possibilities concerning the output list. However, capturing dependencies between outcomes of clustering decisions is done through partial materialization of possible worlds, which results in less compact representation. Such representation is not appropriate in case of highly correlated decisions (e.g., in case of parameterized clustering where all clustering decisions are correlated through their dependency on the algorithm parameter).

In [28], the K-means clustering algorithm is modified to work on objects with uncertain distance metric. The proposed algorithm uses expected distances between objects and clusters centroid to capture distance uncertainty. In contrast, we assume that similarities between pairs of records are deterministic, while uncertainty emerges from the inability to identify the optimal clustering of records based on their similarities.

## 8. CONCLUSION

In this paper, we introduced a novel approach to address uncertainty in duplicate detection and model all possible repairs corresponding to multiple parameter settings of clustering algorithms. We introduced efficient methods to construct a representation of possible repairs by modifying hierarchical clustering algorithms. We showed how to support relational operations and proposed new probabilistic query types, which are not possible under current deterministic approaches. We conducted an experimental study to illustrate the efficiency of our algorithms in various configurations.

## 9. REFERENCES

[1] Bsiness objects, http://www.businessobjects.com.

[2] Oracle data integrator, http://www.oracle.com/technology/products/oracle-data-integrator.

[3] PostgreSQL database system, http://www.postgresql.org.

[4] S. Abiteboul, P. Kanellakis, and G. Grahne. On the Representation and Querying of Sets of Possible Worlds. *SIGMOD Rec.*, 1987.

[5] P. Andritsos, A. Fuxman, and R. J. Miller. Clean answers over dirty databases: A probabilistic approach. In *ICDE*, 2006.

[6] M. Arenas, L. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *PODS*, 1999.

[7] E. Balas and M. W. Padberg. Set partitioning: A survey. *SIAM Review*, 1976.

[8] G. Beskales, M. A. Soliman, I. F. Ilyas, and S. Ben-David. Modeling and querying possible repairs in duplicate detection. *Technical Report CS-2009-15, University of Waterloo*.

[9] I. Bhattacharya and L. Getoor. Collective entity resolution in relational data. *ACM TKDD*, 2007.

[10] S. Chaudhuri, V. Ganti, and R. Motwani. Robust identification of fuzzy duplicates. In *ICDE*, 2005.

[11] S. Chaudhuri, A. D. Sarma, V. Ganti, and R. Kaushik. Leveraging aggregate constraints for deduplication. In *SIGMOD*, 2007.

[12] P. Christen. Probabilistic data generation for deduplication and data linkage. In *IDEAL*, 2005.

[13] P. Christen and T. Churches. Febrl. freely extensible biomedical record linkage, http://datamining.anu.edu.au/projects.

[14] Y. C.Yuan. Multiple imputation for missing data: Concepts and new development. In *the 25th Annual SAS Users Group International Conference*, 2002.

[15] N. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *VLDB J.*, 2007.

[16] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *TKDE*, 2007.

[17] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328), 1969.

[18] G. W. Flake, R. E. Tarjan, and K. Tsioutsiouliklis. Graph clustering and minimum cut trees. *Internet Mathematics*, 2004.

[19] H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C.-A. Saita. Declarative data cleaning: Language, model, and algorithms. In *VLDB*, 2001.

[20] S. Guha, R. Rastogi, and K. Shim. CURE: an efficient clustering algorithm for large databases. In *SIGMOD*, 1998.

[21] J.-C. F. Heiko Mller. Problems, methods, and challenges in comprehensive data cleansing. *Technical Report. Humboldt University Berlin*, 2003.

[22] T. Imieliński and J. Witold Lipski. Incomplete Information in Relational Databases. *J. ACM*, 31(4):761–791, 1984.

[23] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice Hall College Div, 1988.

[24] J. Kubica and A. W. Moore. Probabilistic noise identification and data cleaning. In *ICDM*, 2003.

[25] M.-L. Lee, T. W. Ling, and W. L. Low. IntelliClean: a knowledge-based intelligent data cleaner. In *KDD*, 2000.

[26] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: a tiny aggregation service for ad-hoc sensor networks. *SIGOPS Oper. Syst. Rev.*, 2002.

[27] M. H. Mulry, S. L. Bean, D. M. Bauder, D. Wagner, T. Mule, and R. J. Petroni. Evaluation of estimates of census duplication using administrative records information. *Jour. of Official Statistics*, 2006.

[28] W. K. Ngai, B. Kao, C. K. Chui, R. Cheng, M. Chau, and K. Y. Yip. Efficient clustering of uncertain data. In *ICDM*, 2006.

[29] M. van Keulen, A. de Keijzer, and W. Alink. A probabilistic XML approach to data integration. In *ICDE*, 2005.