Modeling Uncertainty in Duplicate Elimination

George Beskales Mohamed A. Soliman Ihab F. Ilyas
University of Waterloo, Canada
{gbeskale,m2ali,ilyas}@uwaterloo.ca

Technical Report CS-2008-07

Abstract

Real-world databases experience various data quality problems of different causes including heterogeneity of consolidated data sources, imprecision of reading devices, and data entry errors. Existence of duplicate records is a prominent data quality problem. The process of duplicate elimination often involves uncertainty in deciding on the true duplicates. Current tools resolve such uncertainty either through expert intervention, which is not always possible, or by taking destructive decisions that may lead to unrecoverable errors.

In this paper, we approach duplicate elimination from a new perspective treating deduplication procedures as data processing tasks with uncertain outcomes. We propose a complete uncertainty model that compactly encodes the space of clean instances of the input data, and introduce efficient model implementations. We extend our model to capture the behavior of the deduplication process, and allow revising and updating the modeled uncertainty. We apply our model and techniques to state-of-the-art deduplication algorithms to demonstrate the added value of our methods. Our experimental study evaluates the complexity and scalability of our techniques in different configurations.

1 Introduction

Data quality is a key ingredient in successful data analysis and processing. Several sources of noise can negatively affect data quality. Examples include the heterogeneity of integrated data sources, imprecision of reading devices, and data entry errors. Data quality problems include missing values [13], and violation of integrity constraints [19]. Databases that experience such problems are often referred to as unclean databases. Data cleaning is the process of fixing errors and anomalies in an unclean database. Data cleaning is usually an expensive and labor intensive process, which tends to be done once before analyzing or storing the data [18, 21, 24].

One of the major data quality problems is the presence of duplicate records that refer to the same real-world entity. The problem of duplicate elimination is referred to as deduplication, entity resolution [7], or record linkage [15].

In this paper, we approach duplicate elimination from a new perspective treating cleaning procedures as data processing tasks with uncertain outcomes. We present

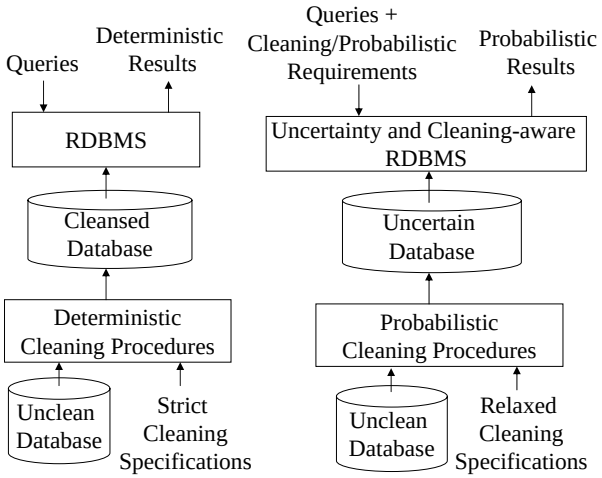


Figure 1: Deterministic Vs. Probabilistic Cleaning

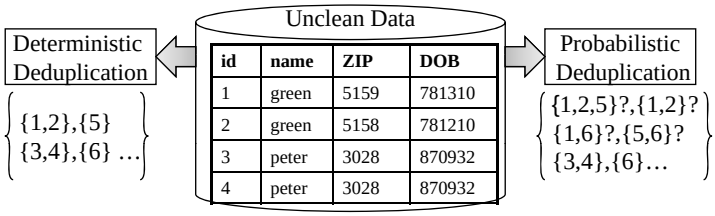


Figure 2: Duplicate Detection Approaches

novel techniques to model the uncertainty involved in deduplication procedures. Our proposed solution extends the implementation of current deduplication techniques by capturing their intermediate cleaning decisions, and the space of possible outcomes.

1.1 Motivation and Challenges

The one-shot deduplication approach, currently adopted by many data cleaning tools, has the following limitations:

- Dealing with Uncertain Cleaning Decisions: The current cleaning approaches are unable to properly handle uncertain cleaning decisions, i.e., deciding whether two records are duplicates. Uncertainty is usually handled through two alternatives. The first alternative is to take arbitrary decisions, which may result in unrecoverable errors, e.g., deciding that two records are duplicates even with weak evidence. The second alternative is to resort to expert intervention, which is not always available and does not scale with large volume of uncertainty in the

unclean data.

- Updatability: The cleansed data produced by current cleaning tools is tightly-coupled with the cleaning specifications used by the cleaning procedures. Changing the cleaning specifications invalidates the cleansed data and requires keeping the original raw data and re-applying the cleaning procedures.

Allowing flexibility in cleaning specifications is imperative to reduce the time and effort spent in re-constructing and maintaining clean data. For example, applications that compute high-level aggregated information might tolerate certain degrees of cleaning errors for the sake of performance. On the other hand, decision making processes usually require highly accurate information, and thus may not tolerate any cleaning errors. Between the two extremes, various degrees of uncertainty may be accepted by different applications.

We address the above challenges by compactly encoding the uncertainty involved in the cleaning procedures, and extending current cleaning systems to support new query types that take both cleaning requirements and uncertainty into consideration. We illustrate our "probabilistic cleaning" approach using Figure 1(b). Our approach uses relaxed cleaning specifications that allow uncertain cleaning decisions, e.g., two records *may* be duplicates, and extends the implementation of cleaning procedures to generate all possible outcomes based on these specifications. The space of cleaning outcomes is maintained using an uncertainty and cleaning-aware RDBMS.

The "probabilistic cleaning" approach builds on and extends the current practices in data cleaning in several ways:

- Capturing all possible cleaning outcomes allows preserving the potentially useful cleaning results without destroying the original unclean data. This enables data processing without expert intervention if uncertain query answers are encountered. Further, it provides experts with a scope of the conflicting/uncertain cleaning decisions that need to be addressed.

- Contrasting and comparing the uncertainty involved in the outputs of multiple cleaning procedures, or the same procedure with multiple specifications. This allow for using and integrating different cleaning tools appropriately in cleaning workflows.

- Enabling new types of cleaning requirements defined on the query output, and hence are independent from the implementation details of underlying cleaning procedures.

- Allowing specifying cleaning requirements as query predicates by pushing the expensive off-line clustering and matching tasks to model building.

Figure 2 gives an example for the output of the deterministic and probabilistic deduplication approaches using census data with duplicate records. Ideally, a clean version of the input table should contain one record per person. Duplicate detection algorithms identify such duplicates by measuring their similarity (more details in Section 2). The output of the duplicate detection algorithm is disjoint groups of record identifiers, where each group denotes a set of duplicate records.

In one-shot deduplication, records are strictly identified as either duplicates or non-duplicates. However, with probabilistic deduplication, this restriction is relaxed to allow for "possible duplicates". The question marks in Figure 2 refer to such possible duplicates. The uncertainty in deciding on the true duplicates is described by an underlying uncertainty model that allows for querying all possible clean outcomes.

1.2 Contributions

We summarize our contributions as follows:

- We propose a complete uncertainty model that compactly encodes the space of clean instances generated by deduplication procedures with uncertain cleaning decisions.
- We extend our model to capture the inter-decision dependencies, and to track the behavior of the cleaning algorithm. We show how our model is flexible in that it can accommodate updates based on new evidences obtained from different sources.
- We introduce new query types that exploit the modeling of all possible clean instances, and are not supported under current one-shot cleaning approaches.
- We apply our model and techniques to the state-of-the-art deduplication algorithms to demonstrate the flexibility of our methods in different settings.

The remainder of this paper is organized as follows. Section 2 discusses the shortcomings of current techniques. In Section 3, we introduce our basic uncertainty model, and we show how to extend it in Section 4. We provide two case studies in Section 5. In Section 6, we discuss querying and updating our uncertainty model. Section 7 gives our experimental study. We briefly describe related works in Section 8. We conclude with final remarks in Section 9.

2 Background and Shortcomings of the Current Techniques

Deduplication algorithms are mainly based on two main operations: *record matching*, and *clustering*. Matching is usually done by measuring record similarity based on a distance measure, e.g., Euclidian distance, edit distance, and Q-grams [15]. The output of record matching can be seen as a graph that connects each pair of similar records with an edge weighted by the strength of their similarity. Clustering aims at grouping records that represent the same entity. Thus, merging each cluster into one record gives a clean duplicate-free version of the input data.

Many deduplication techniques adapt classical clustering algorithms for record deduplication. One example is using the hierarchical clustering algorithm [23, 10, 15, 7] (also called greedy agglomerative clustering). Hierarchical clustering iteratively

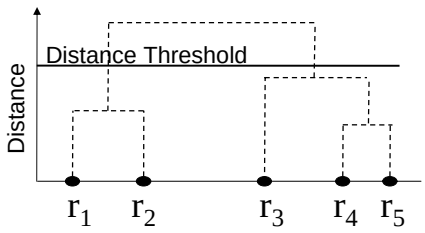


Figure 3: Example of Hierarchical Clustering

groups pairs of records (or record clusters) that are the closest, according to some distance metric. The algorithm terminates when the minimum pairwise distance is greater than a given threshold. For example, in Figure 3, the algorithm clusters r_4 and r_5, r_1 and r_2, and finally r_3 and $\{r_4, r_5\}$. The distance among record clusters is computed in different ways. For example, in single-linkage [20], the distance between two clusters is the distance between the two closest records in the two clusters.

Another family of deduplication algorithms builds on nearest neighbor (NN) techniques to cluster duplicate records. For example, in [9], records are declared duplicates if they represent a *compact set*, i.e., they are mutual nearest neighbors to each other, and they have a *sparse neighborhood*, i.e., the surrounding space is relatively empty.

A common problem in deduplication algorithms is to identify the optimal settings of algorithm parameters to generate the output with the highest accuracy. For example, techniques based on hierarchical clustering, e.g., [7], assume that two records are duplicates if their distance is less than a given threshold. Similarly, nearest neighbor-based techniques, e.g., [9], assume that the sets of nearest neighbor records are duplicates if the sparseness of their neighborhood is above some cut-off threshold. In many cases, using a single threshold value in all the decisions taken by the algorithm does not give the best accuracy. Thus, several proposals assume an *uncertainty region* in the parameter domain, where the deduplication process cannot make a deterministic decision [15, 18, 16, 25]. Hence, records can be classified as 'duplicates', 'non-duplicates', or 'possible duplicates' depending on the computed value of the used metric, e.g., records' distance. Possible duplicates are currently resolved by firing an "exception" that is resolved through expert intervention [18, 22]. When the number of such exceptions is large, or when experts are not available, the cleaning process either stops or is forced to take destructive decisions, e.g., assume that possible duplicates are non-duplicates, to generate a cleansed output. Such destructive decisions are usually *unrecoverable* since undoing any decision requires restarting the whole deduplication process.

3 A Basic Uncertainty Model

In this section, we give a complete uncertainty model that represents all possible clean instances that can be generated by a deduplication process. To illustrate, assume a simple relation $R = \{r_1, r_2, r_3, r_4\}$, where the deduplication process has classified $\{r_1, r_2\}$ as a set of possible duplicates, and $\{r_3, r_4\}$ as a set of duplicates. Hence,

Unclean Relation
Person

Pid	*Name*	*Postcode*	*City*	*State*
P1	Green	5159	Augusta	GA
P2	Paterson	3028	Kansas	MO
P3	Paterson	3028		
P4	Green, Bill	5158		GA
P5	Chuck	2150	Jackson	MS
P6	Green	2150	Augusta	

Cleaning Process

U-Clean Relation
CleanedPerson

Pid	*Name*	*Postcode*	*City*	*State*	L^C
CP1	Green	5159	Augusta	GA	P1
CP2	Green, Bill	5158	Augusta	GA	P1,P4
CP3	Paterson	3028	Kansas	MO	P2,P3
CP4	Chuck	2150	Jackson	MS	P5
CP5	Chuck, Green	2150	Jackson	MS	P5,P6
CP6	Green	2150	Augusta	GA	P1,P6
CP7	Green, Bill	5158		GA	P4
CP8	Green	2150	Augusta		P6

(a) Modeling Uncertain Clean Relations

I_1	I_2	I_3	I_4	I_5
CP1	CP1	CP2	CP2	CP3
CP3	CP3	CP3	CP3	CP4
CP4	CP5	CP4	CP5	CP6
CP7	CP7	CP8		CP7
CP8				

(b) Possible Clean Instances of $CleanedPerson$

Figure 4: Basic Uncertainty Model

we have two possible clean instances: $\{\langle r_3, r_4\rangle, \langle r_1, r_2\rangle\}$ and $\{\langle r_3, r_4\rangle, r_1, r_2\}$, where $\langle r_i, r_j\rangle$ represents the record resulting from merging r_i and r_j. Clean instances have clear analogy to the concept of 'possible worlds' in uncertain databases [6, 14, 4], where possible worlds are all possible database instances originating from tuple-level and/or attribute-level uncertainty.

In general, the number of possible clean instances is exponential in the number of possible duplicates. Hence, explicit generation and storage of such space is not an option. Lineage (a mechanism for tracking the sources of data items) has been used in building complete uncertainty models that can describe arbitrary input instances [6]. This motivates using lineage as a tool to describe the space of possible clean instances. Note that lineage in our settings is 'cleaning-oriented', and hence it has special interpretations that originate from the problem semantics, as we discuss below.

Definition 1 Duplicate Set. *For an unclean relation R, a duplicate set, denoted D, is a set of records in R identified by the cleaning process as duplicates.* □

For each record $r \in R$ that does not belong to any duplicate set, we define a singleton duplicate set $\{r\}$.

Definition 2 Uncertain Clean (U-Clean) Relation. *For an unclean relation R, the corresponding U-Clean relation R^c is a set of records created by merging each duplicate set $D \subseteq R$ into a representative record. Records of R^c are called c-records.* □

A U-Clean relation R^c compactly encodes multiple clean instances of R, which are defined as follow:

Definition 3 Clean Instance. *For a U-Clean relation R^c, a clean instance is a set of records $I \subseteq R^c$ that is (1)* consistent*: records in I are c-records of disjoint duplicate sets; and (2)* maximal*: the union of duplicate sets corresponding to c-records in I is equal to the unclean relation R.* □

Based on the above definitions, we next define our basic uncertainty model to encode the space of clean instances.

Definition 4 Uncertainty Model. *For a U-Clean relation R^c, let the* cleaning lineage *of R^c, denoted by L^c, be a mapping of each c-record $r \in R^c$ to its duplicate set $D \subseteq R$. The uncertainty model is the pair (R^c, L^c).* □

We illustrate the above definitions using the example given in Figure 4. Figure 4(a) shows the components of our uncertainty model for the unclean relation *Person* which contains census data. A cleaning process identifies duplicate records, and generates the corresponding U-Clean relation *CleanedPerson*. The cleaning lineage associates each c-record in the U-Clean relation with its corresponding duplicate set. For example, the c-record $CP2$ corresponds to the duplicate set $\{P1, P4\}$. The *cleaning lineage* can be simply modeled as an additional column in the U-Clean relation, allowing for simple identification of valid clean instances. For example, in Figure 4(a), the c-records $CP1$ and $CP2$ cannot co-exist in the same clean instance due to their intersecting cleaning lineage.

Figure 4(b) shows the space of clean instances $\{I_1, I_2, I_3, I_4, I_5\}$ of the U-Clean relation *CleanedPerson*. Each instance defines a possible partition of the unclean relation. In the worst case, the number of such partitions is exponential in $|R^c|$ (by correspondence to the problem of set partitioning [5]).

An important property in uncertainty models is *completeness*, which is the model ability to capture an arbitrary set of possible instances. The importance of model completeness, in our context, is to cover a wide class of cleaning algorithms, whose output is representable as a set of possible clean instances, without being tied to the low level details of each algorithm. We show that our model is *complete* with respect to the space of possible clean instances.

Theorem 1 *Given an arbitrary set of clean instances $\mathcal{I} = \{I_1, \ldots, I_n\}$, we can always construct a U-Clean relation R^c along with its cleaning lineage L^c, such that the set of clean instances of R^c is the same as $\mathcal{I}$.* □

`Proof.` Each input instance $I_j \in \mathcal{I}$ gives a different partition of the same unclean relation R. That is, each input instance is a set of non-intersecting duplicate sets whose union is R. We extend R to be $\acute{R}$ by adding to R a *dummy* (empty) record λ_i corresponding to each distinct duplicate set D_i appearing in one or more instances in $\mathcal{I}$. We use these dummy records to control the clean instances generated from the U-Clean relation R^c in order to match $\mathcal{I}$. The U-Clean relation R^c is constructed by the following procedure:

1. For each duplicate set D_i appearing in one or more input instances, add to R^c a c-record C_i whose lineage is D_i. We call C_i the c-record corresponding to D_i, and equivalently, we call D_i the duplicate set corresponding to C_i.

2. For each c-record $C_i \in R^c$, add identifier of a dummy record λ_i to the lineage of C_i.

3. For each input instance $I_j \in \mathcal{I}$, add to R^c a *dummy* c-record a_j whose lineage is the set of λ_i's not included in the lineage of any c-record corresponding to any duplicate set in I_j.

Let $\acute{\mathcal{I}}$ be the set of clean instances generated from the constructed relation R^c according to Definition 3). We show the one-to-one correspondence of $\acute{\mathcal{I}}$ and $\mathcal{I}$ by proving that: (i) for any input instance $I_j \in \mathcal{I}$, duplicate sets in I_j correspond to non-dummy c-records in one clean instance in $\acute{\mathcal{I}}$; and (ii) for any clean instance $\acute{I}_j \in \acute{\mathcal{I}}$, non-dummy c-records in $\acute{I}_j$ correspond to duplicate sets in one input instance in $\mathcal{I}$.

We prove (i) as follows. Let $\acute{I}_j$ be the set of c-records in R^c corresponding to duplicate sets in I_j in addition to the dummy c-record a_j. The instance $\acute{I}_j$ is a clean instance of R^c because: (1) the lineage of non-dummy c-records in $\acute{I}_j$ is non-intersecting; (2) the lineage of a_j is disjoint with the lineage of all other records in $\acute{I}_j$; and (3) the c-records in $\acute{I}_j$ correspond to duplicate sets whose union gives the unclean relation $\acute{R}$. Hence, according to Definition 3, $\acute{I}_j$ is both consistent and maximal, and $\acute{I}_j$ is thus a clean instance of R^c. The instance I_j can be simply obtained from $\acute{I}_j$ by removing a_j, and removing λ_i's from the lineage of non-dummy c-records in $\acute{I}_j$

We prove (ii) by contradiction. Assume that there does not exist an input clean instance $I^* \in \mathcal{I}$ that contains duplicate sets corresponding to non-dummy c-records in a clean instance $\acute{I}_j \in \acute{\mathcal{I}}$. Since $\acute{I}_j$ is maximal, the c-records in $\acute{I}_j$ must correspond to duplicate sets whose union gives the unclean relation $\acute{R}$. Let $\mathcal{L}$ be the set of λ_i's that are not included in the lineage of any non-dummy c-record in $\acute{I}_j$. If $\mathcal{L}$ is empty, then all c-records in R^c have non-intersecting lineage and included in $\acute{I}_j$. Consequently, there is exactly one clean instance in $\mathcal{I}$ containing duplicate sets corresponding to all non-dummy c-records in $\acute{I}_j$.

If $\mathcal{L}$ is non-empty, then based on our initial assumption that the instance I^* does not exist in $\mathcal{I}$, there is no dummy c-record in R^c with lineage $\mathcal{L}$. This is due to the way dummy c-records are constructed (step (3) in the construction procedure of R^c). Thus, no dummy c-record in R^c can be a member of $\acute{I}_j$. However, this means that $\acute{I}_j$ is non-maximal, a contradiction. □

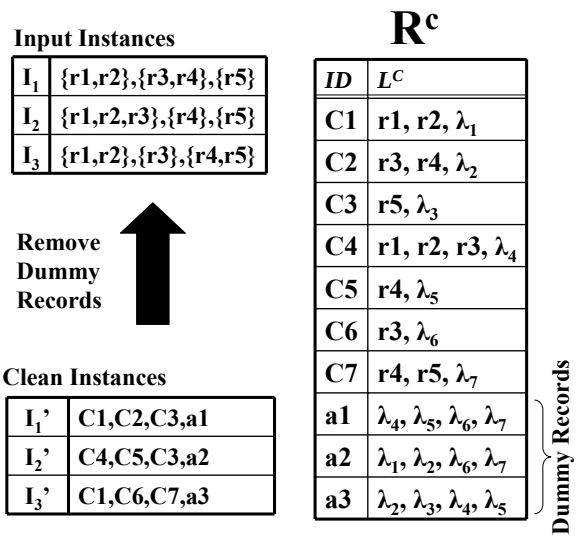


Figure 5: Modeling Arbitrary Input Instances

The intuition of the proof of Theorem 1 is that we can construct the relation R^c by modifying the cleaning lineage to suppress any possible instance that is not included in the given set of input instances $\mathcal{I}$.

We illustrate the completeness of our model using the example depicted by Figure 5 which shows three input instances, I_1, I_2, and I_3, each is represented as a set of duplicate sets. We construct the U-Clean relation R^c by following the procedure given in the proof of Theorem 1, where we create a c-record C_i corresponding to each unique duplicate set, extend the lineage of C_i with the additional identifier λ_i, and finally create three dummy c-records, a_1, a_2, and a_3, each corresponding to one of the input instances. It can be verified that no clean instance can be generated from the constructed R^c other than the shown $\acute{I}_1$, $\acute{I}_2$, and $\acute{I}_3$ instances. For example, although the instance $\{C4, C7\}$ is a clean instance with respect to the original lineage keys $\{r1, r2, r3, r4, r5\}$, the added λ_i's make this instance non-maximal. Further, the dummy c-records prohibit creating any clean instance containing both $C4$ and $C7$. For any clean instance of R^c, we can recover the corresponding input instance (I_1, I_2, or I_3) by simply removing the dummy records.

We note that our basic model is similar in principle to the ULDB model [6], which is intended for general uncertain relations. The ULDB model is based on x-relations and lineage. In an x-relation, a tuple is a set of exclusive alternatives that are non-overlapping with the alternatives of other tuples. However, in our settings, c-records correspond to arbitrary (possibly intersecting) duplicate sets, and hence they cannot be described as x-tuples with non-overlapping alternatives.

Although our basic uncertainty model is complete, it has two main limitations:

- *Record Dependencies:* Maintaining lineage of c-records in the U-Clean relation

does not allow encoding record correlations beyond whether records co-exist in the same clean instance or not. For example, the c-records of the duplicate sets $\{r_1, r_2\}$ and $\{r_3, r_4\}$ co-exist in the same clean instances, however, they may be correlated through the decision that r_1 and r_3 are possible duplicates. That is, the duplicate sets $\{r_1, r_2\}$ and $\{r_3, r_4\}$ requires the non-existence of the duplicate set $\{r_1, r_3\}$.

- *Model Updatability:* The model encodes cleaning lineage as flat sets of record identifiers. Updating the model by overriding uncertain cleaning decisions, e.g., through expert intervention, is not possible since intermediate cleaning decisions are not captured.

In Section 4, we address these limitations by constructing a model that captures the intermediate cleaning decisions.

4 Extending the Model with Uncertain Decisions

The uncertainty model described in Section 3 is viewed as a compact representation of possible outputs of the cleaning process. In this section, we build on this model by constructing a probability space that maps each clean instance to a possible execution path of the deduplication process. We discuss this mapping in Section 4.1. We discuss the representation of the probability space in Section 4.2, followed by a description of model construction in Section 4.3.

4.1 Mapping Clean Instances to Execution Paths

Due to uncertainty in duplicate detection, an uncertain deduplication process has a set of possible execution paths, in contrast to the single execution path of its deterministic counterpart. To illustrate, consider applying the single-linkage hierarchical clustering on a relation $R = \{r_1, r_2, r_3\}$, where the distance between r_1 and r_2 is equal to 7, and the distance between r_2 and r_3 is equal to 3. Using a deterministic distance threshold $\tau = 6$, the algorithm will declare r_2 and r_3 as duplicates, and will declare $\{r_2, r_3\}$ and r_1 as non-duplicates. Thus, the algorithm will follow the execution path that is highlighted in bold in Figure 6. The distance between each pair of clusters is shown above the oval representing that pair.

Alternatively, an uncertain deduplication process defines a range of thresholds $[\tau_1, \tau_2]$, such that any pair of records/record clusters whose distance falls within that range is a possible duplicate. Figure 6 shows all possible execution paths of such process. The outcome of each path is a possible clean instance of the input relation, where each instance takes a (Yes/No) decision on each possible duplicate set. For example, the instance I_2 assumes that r_2 and r_3 are duplicates, while the instance I_5 assumes that they are non-duplicates. Each execution path maps to exactly one clean instance. Furthermore, the same c-record can be generated by different execution paths, e.g., $\langle r_1, r_2, r_3 \rangle$ is generated in both clean instances I_1 and I_3.

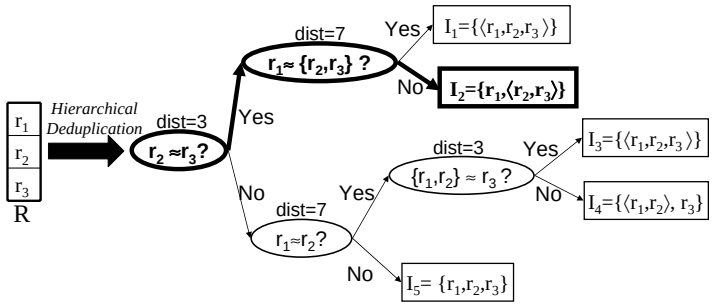


Figure 6: Execution Paths of Single-Linkage Deduplication

Each execution path can be seen as a Boolean assignment to the uncertain decisions. Hence, the joint distribution on these assignments gives us a probability space whose elements are equivalent to the set of clean instances.

Different deduplication algorithms generate different execution paths for the same input data. The reason is that each algorithm defines the order and the criteria of record matching differently. For instance, in our hierarchical deduplication example, merging r_1 and r_2 in a specific execution path rules out the possibility of r_1 being merged with any other record in the same execution path since $\{r_1, r_2\}$ is now a cluster. However, there exist general restrictions that hold for any algorithm. For example, any clean instance cannot have conflicting c-records, e.g., $\langle r_1, r_2 \rangle$ and $\langle r_1, r_3 \rangle$.

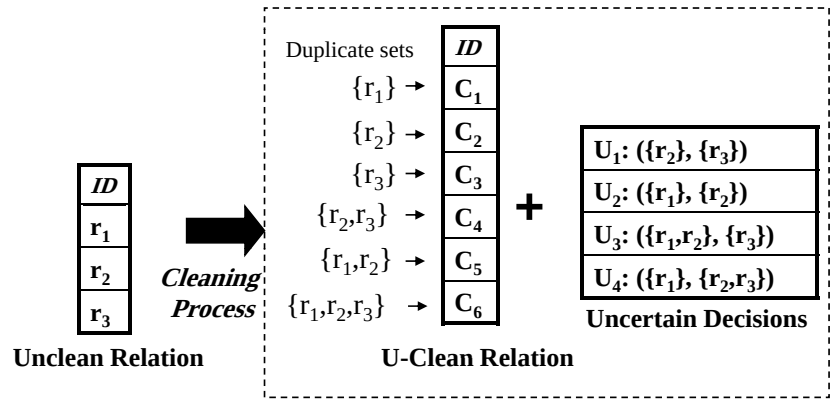
We thus emphasize that we are not after modeling the global space of possible clean instances, i.e., all possible partitions of the input relation under any algorithm, but we are after modeling the clean instances for a given algorithm. Hence, our focus is on providing the machinery that can be used to probabilistically model the uncertainty in a wide class of deduplication algorithms.

4.2 Modeling Uncertain Decisions

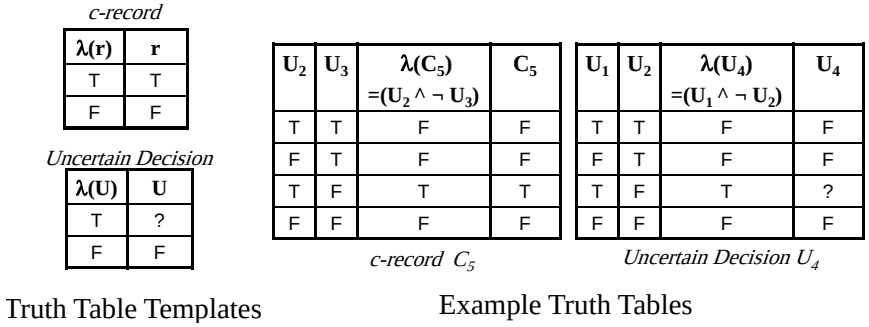
Since the execution paths of a given algorithm cannot be enumerated due to their exponential growth, we show how to concisely represent the probability space using additional constructs. We define *Uncertain Cleaning Decision* as follows.

Definition 5 Uncertain Cleaning Decision. *An uncertain cleaning decision $U(D_1, ..., D_m)$, where $D_i, 1 \leq i \leq m$, are duplicate sets, evaluates the possibility that records in $\bigcup_{i=1}^{m} D_i$ are duplicates. $U(D_1, ..., D_m)$ has two possible outcomes: `True`, where a c-record corresponding to the duplicate set $\bigcup_{i=1}^{m} D_i$ replaces c-records corresponding to D_1 through D_m; and `False`, where $D_1, ..., D_m$ represent separate entities and hence left unmerged.* □

Each uncertain decision divides the space of clean instances into two sets, corresponding to the two possible outcomes of the decision. All possible clean instances corresponding to the `True` outcome of a decision $U(D_1, ..., D_m)$ contain a c-record r corresponding to $\bigcup_{i=1}^{m} D_i$ (or further merge r with other c-records). On the other



(a) Uncertain Decisions Generated by the Cleaning Process



(b) Conditional Dependencies

Figure 7: Modeling Dependencies of Uncertain Decisions

hand, all possible clean instances corresponding to the `False` outcome do not contain a c-record for $\bigcup_{i=1}^{m} D_i$. For example in Figure 6, $U(\{r_2, r_3\}, \{r_1\})$ evaluates the possibility that $\{r_1, r_2, r_3\}$ is a duplicate set, which generates a c-record $\langle r_1, r_2, r_3 \rangle$, or alternatively, the duplicate sets $\{r_2, r_3\}$ and $\{r_1\}$ are left unmerged, and thus become candidates to merge with other c-records.

Assume that the cleaning process has generated the c-record r for the duplicate set $\{r_1, r_2, r_3\}$. Although the model in Section 3 is capable of identifying the possible clean instances containing r, it does not capture how r is actually generated. For example, r can be generated in various ways using hierarchical deduplication: $U(\{r_1\}, \{r_2\})$ followed by $U(\{r_3\}, \{r_1, r_2\})$, or $U(\{r_2\}, \{r_3\})$ followed by $U(\{r_1\}, \{r_2, r_3\})$. Furthermore, each one of these decisions depends on a set of other decisions. For example, merging r_3 with $\{r_1, r_2\}$ assumes that $\{r_1, r_2\}$ is declared as a duplicate set, i.e., $U(\{r_1\}, \{r_2\})$ is `True`.

We view an uncertain decision U as a Boolean random variable that is equal to `True` iff: (1) U is considered in a randomly chosen execution path of the deduplication process, and (2) its outcome is `True` in that path. Since the existence of c-records depends on the outcomes of the uncertain decisions, c-records are also viewed

as Boolean random variables, reflecting the event that they appear in the output generated by a randomly chosen execution path. In the following, we use 'c-record' to refer to 'c-record variable', and 'decision' to refer to 'decision variable', when it is clear from the context.

Constructing the probability space on execution paths requires identifying the dependencies among their components. We identify two types of such dependencies: (1) dependencies of c-records on uncertain decisions, these dependencies are common for all deduplication algorithms; and (2) dependencies among uncertain decisions, these dependencies are algorithm-specific. We describe the first type of dependencies and give a general overview on the second type of dependencies, with concrete examples given in Section 5.

• **Record-Decision Dependencies.** Let $\mathcal{U}$ be the set of all uncertain decisions. For a c-record r with a corresponding duplicate set D, let $\mathcal{U}_r^+$ be the set of decisions that generate r, and $\mathcal{U}_r^-$ be the set of decisions that results in merging D with other records. That is,

$$\mathcal{U}_r^+ = \{U(D_1, ..., D_m) \in \mathcal{U} : \bigcup_{i=1}^{m} D_i = D\} \tag{1}$$

$$\mathcal{U}_r^- = \{U(D_1, ..., D_m) \in \mathcal{U} : D \subset \bigcup_{i=1}^{m} D_i\} \tag{2}$$

The existence of a c-record r is conditioned on the event that a randomly chosen execution path assigns a `True` outcome to a decision that generates r, and `False` to all decisions that merge r with other records. Hence r is `True` iff at least one decision in $\mathcal{U}_r^+$ is `True`, and all decisions in $\mathcal{U}_r^-$ are `False`. We formulate such condition, denoted as $\lambda(r)$, as follows:

$$\lambda(r) = \quad (\bigvee_{U \in \mathcal{U}_r^+} U) \wedge (\bigwedge_{U \in \mathcal{U}_r^-} \neg U) \tag{3}$$

Based on Equation 3, we have $(\lambda(r) \rightarrow r)$, and $(\neg\lambda(r) \rightarrow \neg r)$, which means that $\lambda(r)$ is necessary and sufficient condition for the existence of r. For example, Figure 7(a) shows an example U-Clean relation and the set of uncertain decisions generated by a cleaning process. Figure 7(b) shows a truth table template listing the truth value of a c-record r given each possible truth assignment of $\lambda(r)$. Because of the equivalence of r and $\lambda(r)$, the truth values are always equal. Figure 7(b) also gives an example truth table for the c-record C_5. Based on the given set of uncertain decisions, $\lambda(C_5) = (U_2) \wedge (\neg U_3)$. The truth table shows all possible assignments of U_2 and U_3 along with the corresponding C_5 value.

If both $\mathcal{U}_r^+$ and $\mathcal{U}_r^-$ are empty, then existence of c-record r is unconditioned, i.e., $\lambda(r) = True$. In other words, the c-record r would exist in all possible clean instances.

• **Decision-Decision Dependencies.** An uncertain decision U depends on previous decisions considered by the deduplication algorithm before considering U. These previous decisions are classified as *prerequisite decisions*, denoted by $\mathcal{U}_U^+$, that must be `True` in order to consider U, and *conflicting decisions*, denoted by $\mathcal{U}_U^-$, that have to be `False` if $U =$ `True`. The precondition of having U equal to `True` is encoded using an expression $\lambda(U)$, defined as follows:

$$\lambda(U) = f(\mathcal{U}_U^+, \mathcal{U}_U^-) \tag{4}$$

where f is an algorithm-specific logical function. If the sets $\mathcal{U}_U^+$ and $\mathcal{U}_U^-$ are empty, then the decision variable U has no prerequisites or conflicts, i.e., $\lambda(U) = True$. The condition $\lambda(U)$ is necessary, but not sufficient, for U to be `True`.

Figure 7(b) shows a truth table template for the possible truth assignments of a decision variable U given the expression $\lambda(U)$. Given that $\lambda(U) =$ `True`, the variable U can be either `True` or `False` representing possible decision outcomes. However, given that $\lambda(U) =$ `False`, the variable U must be `False`. Figure 7(b) gives an example truth table for the decision variable U_4, where $\lambda(U_4) = (U_1) \wedge (\neg U_2)$ (based on the dependencies of the hierarchical deduplication discussed in Section 5). The truth table shows all possible assignments of U_1 and U_2 along with the corresponding U_4 value.

4.3 Model Construction

The expressions $\lambda(r)$ and $\lambda(U)$, given in Section 4.2, encode the necessary conditions for generating a c-record r, or an uncertain cleaning decision U, respectively. Hence, the random variable representing r is conditionally independent from all other variables given $\lambda(r)$, and, similarly, U is conditionally independent from all other variables given $\lambda(U)$. Conditional independence allows maintaining a small set of dependencies while still being able to derive all possible dependencies, similar to Bayesian networks. We introduce a graphical representation of our dependency model based on the concept of conditional independence. Specifically, our extended model is defined as follows:

Definition 6 Extended Uncertainty Model. *For a U-Clean relation R^c, and a set of uncertain cleaning decisions $\mathcal{U}$, let $G(V, E)$ be a DAG, denoted as* `Dependency-Graph`*, where the set of vertices $G.V$ includes a node per each c-record $r \in R^c$ and a node per each decision $U \in \mathcal{U}$, while the set of edges $G.E$ encodes the expressions $\lambda(r)$ and $\lambda(U)$ for each $r \in R^c$, and each $U \in \mathcal{U}$. The extended uncertainty model is the triple $(R^c, \mathcal{U}, G)$.* □

Encoding the condition $\lambda(r)$ in G is done by labeling all edges (U, r) for $U \in \mathcal{U}_r^+$ with $'T'$, and labeling all edges (U, r) for $U \in \mathcal{U}_r^-$ with $'F'$. Hence, for a given c-record r, we can construct the expression $\lambda(r)$ using the edge labels. Encoding $\lambda(U)$ is done differently according to the underlying deduplication algorithm as we show in Section 5.

The extended uncertainty model given in Definition 6 still encodes the entire cleaning lineage of c-records. The cleaning lineage of a c-record r is obtained from any of the decision nodes that generate r. Hence, the model is still complete. However, the model further maintains dependency information that allows additional functionalities that are not supported by the lineage-based model. Specifically, we describe how to use the dependencies captured by our model in revising cleaning decisions and in computing the probabilities of cleaning outcomes in Section 6.

Algorithm 1 describes model construction. The algorithm initializes the U-Clean relation R^c and the nodes of the `Dependency-Graph` G with the set of base records

Algorithm 1 `Uncertain-Dedup`(R)

Require: R: Unclean relation

1: $R^c \leftarrow R$
2: $\mathcal{U} \leftarrow \phi$
3: Create a decision queue Q initialized with all uncertain decisions with empty prerequisites (Section 5)
4: Create $G = (V, E)$, with $V =$ records in R^c, and $E = \phi$
5: **while** (Q is not empty) **do**
6: Remove first decision $U_i(D_1, ..., D_m)$ from Q
7: **if** (U_i's outcome is deterministically `True`) **then**
8: Construct a c-record r_i corresponding to $\bigcup_{j=1}^{m} D_j$
9: Add r_i to R^c, and create a node r_i in $G.V$
10: Remove from R^c and $G.V$ all c-records corresponding to any duplicate set $D_1, ..., D_m$
11: **else if** (U_i's outcome is a uncertain) **then**
12: Add U_i to $\mathcal{U}$, and $G.V$
13: Add r_i to R^c, and create a node r_i in $G.V$
14: Update $G.E$ based on $\lambda(r_i)$ and $\lambda(U_i)$ (Section 5)
15: Generate new decisions based on current R^c and add them to Q (Section 5)
16: **end if**
17: **end while**
18: **return** $(R^c, \mathcal{U}, G)$

in the unclean relation R. Initially, the set of edges $G.E$ is empty. The algorithm maintains a queue of cleaning decisions, ordered by the sequence in which they should be considered by the deduplication algorithm (e.g., closest-pairs first in case of hierarchical deduplication). For each examined decision $U_i(D_1, ..., D_m)$, the corresponding c-record r_i is created and added to R^c and $G.V$. If the outcome of U_i is deterministically `True`, we remove all c-records corresponding to any of the duplicate sets $D_1, ..., D_m$ from both R^c and $G.V$. On the other hand, if U_i is uncertain, we keep all c-records corresponding to the duplicate sets $D_1, ..., D_m$, and insert U_i into the set of uncertain decisions $\mathcal{U}$ and $G.V$. We also update $G.E$ to capture the new dependencies. The algorithm terminates when Q is empty, where it returns $(R^c, \mathcal{U}, G)$.

Algorithm 1 is a generic algorithm that is used to construct our uncertainty model. The implementation details are defined based on the adopted deduplication approach.

To support a specific deduplication technique, we define two main tasks: (1) modeling uncertainty in clustering as a set of dependent deduplication decisions, and (2) defining dependency among decisions as necessary and/or sufficient conditions for decision checking and c-records creation. The complexity of these tasks depends on the specific deduplication approach. In Section 5, we give two case studies for hierarchical deduplication and NN-based deduplication.

5 Case Studies

In the this section, we show how to use our techniques to model two deduplication algorithms: hierarchical deduplication, which has been widely used in related works (e.g., [23, 10, 15, 7]) and NN-based clustering [9].

5.1 Hierarchical Deduplication

In hierarchial clustering, two clusters are considered duplicates, and hence merged, if their distance is less that a threshold τ. As we showed in Section 4.1, uncertainty emerges from the fact that no single threshold could yield a perfect separation between duplicates and non-duplicates [9]. Thus, in uncertain hierarchical deduplication, we use a range of possible thresholds $[\tau_l, \tau_u]$ such that records/clusters with distances below τ_l are duplicates, records/clusters with distances above τ_u are non-duplicates, and records/clusters with distances in $[\tau_l, \tau_u]$ are possible duplicates.

• **Uncertain Decisions and Dependencies.** Each step in hierarchical deduplication merges a pair of records/clusters. Hence, an uncertain decision U takes the form $U(D_1, D_2)$, where D_1 and D_2 are duplicate sets. Based on this constraint, we identify the following decision dependencies. $U(D_1, D_2)$ implies that: (1) both D_1 and D_2 are singletons or already declared as duplicate sets by some previous decision; and (2) all previous decision U_i that has either D_1 or D_2 as a parameter are equal to `False`. We formulate these dependencies using the following logical expressions:

$$\begin{array}{l} U(D_1, D_2) \rightarrow \\ (|D_1| = 1 \vee \exists U_1(D_{11}, D_{12}) \in \mathcal{U}(D_{11} \cup D_{12} = D_1 \wedge U_1)) \wedge \\ (|D_2| = 1 \vee \exists U_2(D_{21}, D_{22}) \in \mathcal{U}(D_{21} \cup D_{22} = D_2 \wedge U_2)) \end{array} \tag{5}$$

where $|D_i|$ indicate the cardinality of a duplicate set D_i, and

$$U(D_1, D_2) \rightarrow \forall U_i(D_1, D_i), U_j(D_j, D_2) \in \mathcal{U}(\neg U_i \wedge \neg U_j) \tag{6}$$

• **Graphical Dependency Structure.** Based on Equations 5 and 6, the prerequisites of $U(D_1, D_2)$ are encoded as follows:

$\lambda(U(D_1, D_2)) =$

$$(\bigvee_{U_i \in \mathcal{U}^+_{D_1}} U_i) \wedge (\bigvee_{U_j \in \mathcal{U}^+_{D_2}} U_j) \wedge (\bigwedge_{U_k \in \mathcal{U}^-_{(D_1, D_2)}} \neg U_k) \tag{7}$$

where $\mathcal{U}^+_{D_i}$ is the set of decisions that generate D_i in U, while $\mathcal{U}^-_{(D_1, D_2)}$ is the set of decisions that merge D_1 or D_2 with any other duplicate set.

Figure 8 shows how to use our `Dependency-Graph` to represent the sequence of uncertain decisions in the hierarchical deduplication example in Figure 6.

We encode the expression $\lambda(U)$ as follows. A decision node $U_i(D_{i1}, D_{i2})$ is connected to another decision node $U_j(D_{j1}, D_{j2})$ in the following cases:

- If $U_i \in \mathcal{U}^+_{D_{j1}}$. Edge is labeled $'T'_1$, e.g., (U_2, U_3).
- If $U_i \in \mathcal{U}^+_{D_{j2}}$. Edge is labeled $'T'_2$, e.g., (U_1, U_4).

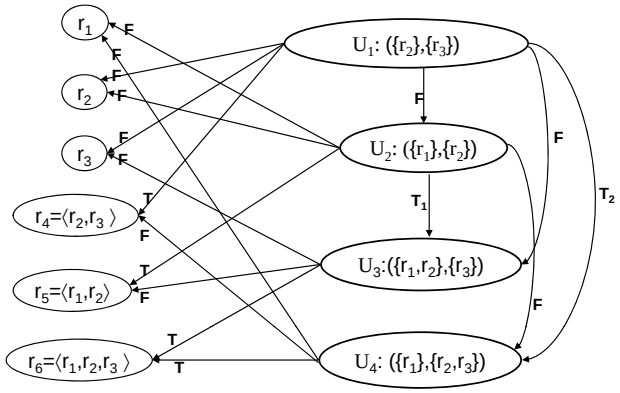


Figure 8: `Dependency-Graph` for Hierarchical Deduplication

- If $U_i \in \mathcal{U}^{-}_{(D_{j1},D_{j2})}$. Edge is labeled $'F'$, e.g., (U_1, U_2).

For a node $U(D_1, D_2)$, labeling an edge (U, U_i) with T_1 denotes that U_i must be `True` for D_1 to exist. Similarly, labeling (U, U_i) with T_2 denotes that U_i must be `True` for D_2 to exist. Labeling (U, U_i) with F denotes that U_i must be `False` for U to be `True`.

• **Quantifying Uncertainty in Decisions.** A probability density function (PDF) $f_\tau(.)$ is assumed on the threshold range $[\tau_l, \tau_u]$ to represent the uncertainty on the correct threshold. Such PDF can be obtained by analyzing the underlying data, or using domain experience. For example, in [25], a range of thresholds is obtained such that the rates of false positives and false negatives are minimized. If no other evidences are available, we can assume a uniform distribution over the threshold range. We use the threshold PDF as follows. Given a decision $U_i(D_1, D_2)$, the distance d_i between the duplicate sets D_1 and D_2 is mapped to the threshold range, such that U_i=`True` at every threshold $\tau > d_i$. Since each decision is dependent on a set of prerequisites, the probability computed using $f_\tau(.)$ is actually conditioned on the event that the decision's prerequisites are already satisfied. That is,

$$\Pr(U_i|\lambda(U_i) = \texttt{True}) = \int_{d_i}^{\tau_u} f_\tau(x)\, dx \tag{8}$$

• **Constructing the Model.** Decisions are created in hierarchical deduplication iteratively. The initial decision inserted in the priority queue Q (line 3 in Algorithm 1) is the closet pair of records. Every time a new cluster is created, we search for the current closest clusters and we insert their matching decision in Q (line 15). We create edges (line 14) among uncertain decisions as described in this section, and between uncertain decisions and c-records as described in Section 4.3.

5.2 NN-based Deduplication

The NN-based deduplication algorithm in [9] defines a duplicate set as a *compact set* that has a *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 the number of records whose distance to r is smaller than double the distance between r and its nearest neighbor. A set S has a sparse neighborhood if $(Agg_{r \in S}\ ng(r) < C)$, where Agg is an aggregate function such as max or avg, and C is a cut-off threshold for neighborhood sparseness. An uncertain NN-based deduplication algorithm is the same as the previous algorithm with the exception that a range of cut-off thresholds $[C_l, C_u]$ is used to express the uncertainty in determining neighborhood sparseness. Such potential uncertainty is also noted in [9], where it is shown that the accuracy of duplicate detection improves by tuning C.

• **Uncertain Decisions and Dependencies.** Decisions in NN-based deduplication have the form $U(D_1, ..., D_m)$, where each duplicate set D_i is a singleton set representing a base record $r \in R$. Let $S(U) = \cup_{j=1}^{m} D_j$. For each compact set, the uncertain NN-based algorithm declares $S(U)$ as a duplicate set if its aggregated neighborhood growth is less than C_l, a non-duplicate set if its neighborhood growth is greater than C_l, or a possible duplicate set if its neighborhood growth is in $[C_l, C_u]$.

The NN-based algorithm does not enforce any dependency between decision. While independence between non-intersecting decisions is straightforward, it is necessary to show that independence holds for intersecting decisions. We observe that intersecting compact sets must have containment relationship. That is, $\forall U_i, U_j (S(U_i) \cap S(U_j) = \phi \vee (S(U_i) \subset S(U_j)) \vee (S(U_j) \subset S(U_i)))$. We verify this fact by contradiction. Assume that $\exists U_i, U_j (S(U_i) \cap S(U_i) \neq \phi \wedge S(U_i) - S(U_j) \neq \phi \wedge S(U_j) - S(U_j) \neq \phi)$. Let k_i and k_j denote the cardinality of $S(U_i)$ and $S(U_j)$, respectively, and assume, without loss of generality, that $k_i \leq k_j$. According to the definition of compact sets [9], the $k_i - 1$ nearest neighbours of a records $x \in S(U_i)$ are equal to $S(U_i) - \{x\}$, and similarly, the $k_j - 1$ NNs of $x \in S(U_j)$ are equal to $S(U_j) - \{x\}$. For $x \in S(U_i) \cap S(U_j)$, the $k_i - 1$ NNs of x are not contained in the set of the $k_j - 1$ NNs of x because $S(U_j) - S(U_i) \neq \phi$. This contradicts the fact that $k_i - 1$ NNs of x must be contained in the $k_j - 1$ NNs of x, assuming that the k-NNs of x are uniquely defined, i.e., no ties in distance.

Furthermore, declaring records in a compact set as duplicates does not require its (compact) subsets to be duplicates, according to the algorithm in [9]. Additionally, if two sets S_1 and S_2, $S_1 \in S_2$, are declared as duplicate sets, the algorithm consider all records in the larger set S_2 as duplicates. If follows that arbitrary assignments of decisions outcomes are possible. Thus, the random variables representing the uncertain decisions are independent, i.e., $\lambda(U) = \texttt{True}, \forall U \in \mathcal{U}$, and hence there are no edges connecting decision nodes in G.

• **Quantifying Uncertainty in Decisions.** Similar to our discussion on the uncertain hierarchical deduplication algorithm, we assume a PDF $f_C(.)$ on the range $[C_l, C_u]$ that expresses the uncertainty about the correct sparseness cut-off. The conditional probability of a decision U is the same as its marginal probability, since all decisions have empty prerequisites. This probability is computed as follows:

$$\Pr(U|\lambda(U) = \texttt{True}) = \Pr(U) = \int_{Agg_{r \in S(U)} ng(r)}^{C_u} f_C(x)\, dx \tag{9}$$

• **Constructing the Model.** The queue Q in Algorithm 1 contains decisions corresponding to all possible compact sets. These decisions could be either inserted at the beginning of the algorithm (line 3), or incrementally constructed and inserted into Q (line 15). Decisions are retrieved from Q in any arbitrary order. Edges are only created (line 14) between uncertain decisions and dependent c-records (Section 4.3).

6 Querying and Updating the Uncertainty Model

In this section, we discuss new data cleaning operations that are enabled using our uncertainty model. We describe in Section 6.1 how to compute the marginal probabilities of c-records, since we need it in the rest of this section. In Section 6.2, we discuss updating the uncertainty model using additional evidences. We discuss supporting new types of queries in Section 6.3.

6.1 Computing the Marginal Probabilities of c-records

Based on our probability space, the marginal probability of a c-record r is the summation of the probabilities of all clean instances containing r. We use the conditional independence between nodes in G to compute the probability of each node based on Bayes rule. The marginal probability of a c-record r, denoted $\Pr(r)$, is computed as $\Pr(r) = \Pr(\lambda(r))$. For example in Figure 8, assume the shown matchings have the following conditional probabilities: $\Pr(U_1|\lambda(U_1)) = 0.8$, $\Pr(U_2|\lambda(U_2)) = 0.6$, $\Pr(U_3|\lambda(U_3)) = 0.8$, and $\Pr(U_4|\lambda(U_4)) = 0.6$. The membership probability of the c-record r_1 is computed as follows:

$$\begin{aligned}
&\Pr(r_1) = \Pr(\lambda(r_1)) \\
&= \Pr(\neg U_2 \wedge \neg U_4) \\
&= \Pr(\neg U_4|\neg U_2) \cdot \Pr(\neg U_2) \\
&= \Pr(\neg U_4|\neg U_2, U_1) \cdot \Pr(\neg U_2|U_1) \cdot \Pr(U_1) \\
&+ \Pr(\neg U_4|\neg U_2, \neg U_1) \cdot \Pr(\neg U_2|\neg U_1) \cdot \Pr(\neg U_1) \\
&= (1.0)(0.4)(0.8) + (1.0)(0.4)(0.2) = 0.4
\end{aligned}$$

Similar to computing individual record's probability, we apply Bayes chain rule to compute the joint probability of multiple records.

In general, Bayesian inference is known to be in NP-hard [8]. Approximate inference, e.g., Markov chain Monte Carlo (MCMC) method [2] can be used to provide more efficient computation of records' marginal and joint probabilities. In our model prototype, we experimented with an approximate inference methods based on Gibbs Sampling [2], as we discuss in our experiments.

6.2 Updating the Uncertainty Model

The uncertainty we capture in our model can be updated/revised in different ways:

- *Revising Record Matchings*. An expert may confirm/invalidate record matchings. Such revisions are encoded by the model without re-cleaning the data.

- *Changing the Similarly Measure*. A more elaborate, e.g., domain-specific, similarity measure may be used to re-evaluate uncertain decisions.

- *Tuning the Parameters of the Cleaning Process*. Tuning the parameters of the cleaning process, e.g., changing the range of possible similarity thresholds, results in modifying the outcomes of uncertain decisions.

We show here how to support update operation on our model through revising the outcomes of uncertain cleaning decisions. An uncertain decision U_i is revised by setting its conditional truth value $U_i|(\lambda(U_i) = \texttt{True})$ (represented as '?' in the truth table in Figure 7(b)) to either `True` or `False`. Revising an uncertain decision U_i does not override its pre-conditions $\lambda(U_i)$. For example, revising the uncertain decision $U(\{r_1, r_2\}, \{r_3\})$ affects the matching between $\{r_1, r_2\}$ and r_3, but not between r_1 and r_2. Thus, updating U_i only affects its descendant nodes in the `Dependency-Graph`. We distinguish between two possibilities according to how $U_i|(\lambda(U_i) = \texttt{True})$ is set:

- $U_i|(\lambda(U_i) = \texttt{True})$ is set to `False`. In this case, the variable U_i is `False`, regardless of $\lambda(U_i)$. For example in Figure 8, setting $U_4|(\lambda(U_4) = \texttt{True})$ to `False`, where $\lambda(U_4) = (U_1 \wedge \neg U_2)$, results in $U_4 = False$ regardless of U_1 and U_2.

- $U_i|(\lambda(U_i) = \texttt{True})$ is set to `True`. In this case, U_i becomes equivalent to $\lambda(U_i)$, which means that we cannot rule out any of the two possible outcomes of U_i. For example in Figure 8, setting $U_4|(\lambda(U_4) = \texttt{True})$ to `True` results in having U_4 =`True` only when U_1 =`True` and U_2 =`False`, and having U_4 =`False` otherwise. A special case is when $\lambda(U_i)$ is always `True` (e.g., U_i has no parent nodes in the graph G. In this case, U_i is known to be always `True`. For example, U_1 in Figure 8 has no parents, i.e., $\lambda(U_i)$ =`True`. Thus, setting $U_i|(\lambda(U_i) = \texttt{True})$ to `True` is equivalent to setting U_i =`True`.

Let $Pa(v)$ denote the parents of a node v in our `Dependency-Graph`. When a decision node U_i becomes deterministic (either `True` or `False`), we remove U_i from the `Dependency-Graph` G. A node n that is a child of U_i, i.e., $U_i \in Pa(n)$, may in turn become deterministic in the following two cases:

- n is a decision-node and $\lambda(n)$ becomes `False` based on the value of U_i. In this case, n is set to `False` since its pre-condition $\lambda(n)$ is `False`. For example in Figure 8, setting U_1 =`True` makes $\lambda(U_2) = \neg U_1$ =`False` and subsequently U_2 =`False`.

- n is a record-node and $\lambda(n)$ becomes either `True` or `False`. In this case, n follows the value of $\lambda(n)$ (i.e., $\lambda(n) \rightarrow n$, and $\neg\lambda(n) \rightarrow \neg n$). For example, having U_1 =`True` and U_4 =`False` results in $\lambda(r_4) = U_1 \wedge \neg U_4$ =`True`, and thus r_4 =`True`, and having $\lambda(r_2) = \neg U_1 \wedge \neg U_2$ =`False`, and thus r_2 =`False`.

Algorithm 2 `Update_Decision` (U_i, d)

Require: U_i: Uncertain decision to be revised
Require: d: the new outcome of U_i, either `True` or `False`

```
1: if (d = True ∧ Pa(U_i) = φ) ∨ d = False then
2:    for each child n of U_i do
3:       if n is a decision-node and λ(n) = False then
4:          Update_Decision(n, False)
5:       else if n is a record-node and λ(n) = False then
6:          Remove n from both G and R^c
7:       end if
8:    end for
9:    Remove U_i from 𝒰, and G (and all connected edges)
10: end if
```

We recursively repeat the above update procedure on reaching decision-nodes that become deterministic. The details of the model update procedure, called `Update_Decision`, is given in Algorithm 2. The algorithm checks if the decision variable U_i has a deterministic value. If this is the case, the children nodes of U_i are re-evaluated, and the algorithm is recursively called if any child decision-node becomes deterministic. Record nodes that become `False` result in removing the corresponding c-records from the U-Clean relation R^c and from the graph G. Decision nodes that become either `True` or `False` are removed from the graph G and from $\mathcal{U}$.

For example, in Figure 8, assume that we set $(U_2|\lambda(U_2) = \texttt{True})$ to be `False`. In this case, we remove U_2 and all its connected edges. In addition, we remove the nodes U_3 and r_5 since their values become `False`. Setting U_3 to `False` will not affect any children nodes, i.e., remain uncertain, and thus the algorithm terminates.

6.3 Querying the Uncertainty Model

Modeling uncertainty in deduplication process allows for new types of queries that are not supported using one-shot deduplication approach. In the following, we describe examples of possible queries.

6.3.1 Retrieving α-certain Answers

Certain answers over uncertain/inconsistent databases has been studied extensively (e.g., [17]). Our model allows for quantifying the confidence in existence of c-records in output, which can be used to report c-records that exhibit a minimum degree of certainty. For example, a user can define a probability threshold α such that all c-records with existence probabilities greater than α are reported. We call this type of queries α-certain queries. Using probabilities as predicates necessitates computing the marginal probabilities of c-record. In large dependency graphs, probability computation could be expensive, and thus could be done either off-line, where indexes on the probability value can be used for efficient retrieval, or by using approximate inference techniques [8]. We use the approximate computation of marginal probabilities in our experiments.

Note that the reported c-records do not necessarily represent a valid clean instance. However, it is always possible to compute the probability of co-existence of multiple records in query output using our uncertainty model.

6.3.2 Most Probable Clean Instance

Obtaining the most probable clean instance is equivalent to the problem of finding *Maximum A Posteriori* (MAP) in Bayesian Networks. Finding the exact MAPs or approximating MAPs with a constant ratio bound is NP-hard [1]. However, in some special cases, it is possible to find MAPs more efficiently. For example, in the `Dependency-Graph` generated by the uncertain NN-based deduplication, decision variables are independent. Hence, finding the most probable instance can be done in polynomial time by setting each uncertain decision separately to its most probable outcome. Assignments of c-records variables are then obtained based on the dependencies of c-records on decisions.

In the general case, the most probable instance can be approximated using Monte-Carlo techniques, similar to [26]. The main idea is to iteratively draw samples from the joint probability distribution of uncertain decisions. Iterative generation of samples is efficiently performed by exploiting conditional independence in our `Dependency-graph`.

6.3.3 Extracting Clean Instances

Our model allows efficient extraction of clean instances corresponding to a specific value of cleaning parameters, e.g., similarity thresholds. Hence, the cost of constructing the uncertainty model to compactly encode a wide range of clean instances is amortized over the number of clean instances requested by the user.

Enabling this operation in our model requires keeping additional metadata that allows setting the outcome of each uncertain decision based on the given parameter value. For example, in uncertain hierarchical deduplication, we annotate each uncertain decision $U(D_1, D_2)$ with the computed distance between D_1 and D_2. Similarly, in uncertain NN-based deduplication, we annotate each uncertain decision with the aggregate neighborhood growth of its corresponding compact set. Once all decisions' outcomes are set, it is straightforward to retrieve the set of c-records that belong to the required clean instance. Complexity of extraction of a clean instance is polynomial as it requires a linear scan of uncertain decisions, followed by evaluation of the prerequisites of c-records.

7 Experiments

In this section, we describe the experimental study we conducted to evaluate our techniques.

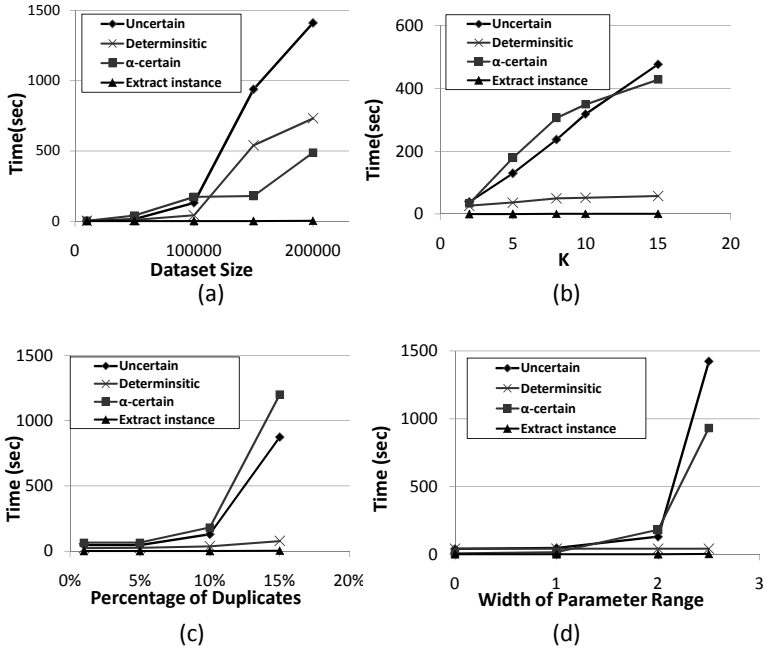


Figure 9: Performance of Hierarchical Deduplication

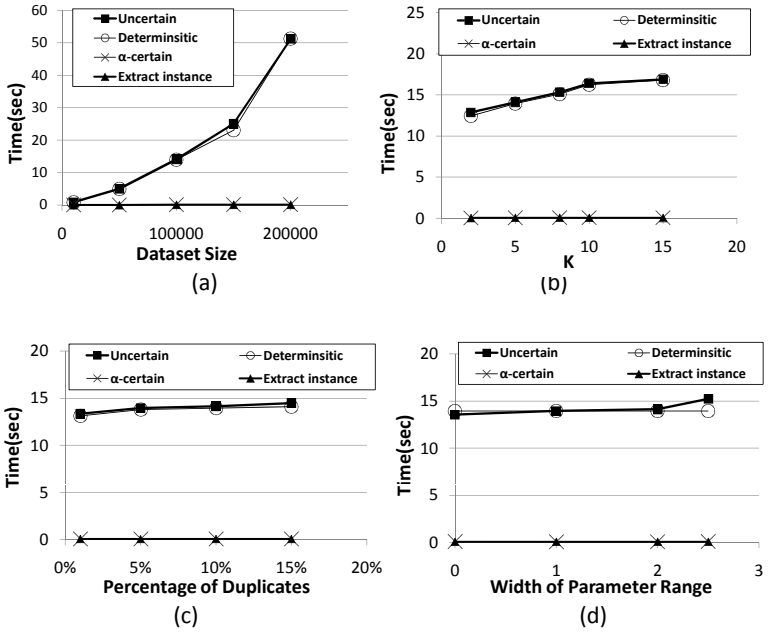


Figure 10: Performance of NN-based Deduplication

7.1 Experiments Setup

All experiments were conducted on a SunFire X4100 server with Dual Core 2.2GHz processor, and 2GB of RAM. We use a synthetic data generator that is part of the Febrl project (Freely Extensible Biomedical Record Linkage) [12]. Datasets generated using Febrl exhibit the content and statistical properties of real-world datasets [11], including the frequency distributions of attribute values, error types, and error positions within attribute values. The distance between records is computed in Febrl based on the similarity between their attributes. The runtime reported in our experiments does not include the time required for computing the distance between records. Our experiments parameters are listed as follows:

- Dataset size: The number of records in the input unclean relation (default is 100,000).
- Maximum records per cluster (K): The maximum number of records allowed per cluster (default is 5).
- Percentage of Duplicates: The percentage of the input relation that represent duplicate records (default is 10%).
- Range of Uncertainty: The width of the parameter range used in deduplication (default is 2, which is 10% of broadest range width).

We implemented hierarchical deduplication and NN-based deduplication (Section 5). We report the running time of (1) uncertain deduplication, (2) deterministic deduplication, (3) extracting a clean instance, and (4) answering α-certain queries (Section 6). We omit running times for obtaining the most probable clean instance as they are almost equal to α-certain queries. We use sampling to compute the marginal probabilities in case of hierarchical deduplication, and we use exact computation in case of NN-based deduplication.

7.2 Performance Results of Hierarchical Deduplication

The running time of uncertain hierarchical deduplication is greater than the deterministic version of the algorithm (Figure 9). This is due to the increased number of decisions and records that are constructed in case of uncertain clustering. However, extracting a clean instance takes less than a second in most cases. Consequently, our approach is more efficient if the user requests multiple clean instances at various values of the cleaning parameters. For example, using our approach to build more than 3 instances under the default configuration consumes less time than performing multiple invocations of the deterministic algorithm.

Effect of Dataset Size. In Figure 9(a), both deterministic and uncertain versions of the algorithm show rapid increase in running time for larger datasets. This is due to the inherent complexity of the hierarchical clustering [20]. The ratio between running times of the deterministic and uncertain versions is almost fixed for all dataset sizes. The running time of α-certain queries is noticeably affected by the increasing size (and complexity) of the `Dependency-Graph`.

Effect of K. As shown in Figure 9(b), the running time of uncertain deduplication shows a linear increase when K grows, while the running time of the deterministic algorithm slightly increases. The reason is that the number of uncertain decisions rapidly increases for larger clusters.

Effect of Percentage of Duplicates. Figure 9(c) shows the running times against the percentage of duplicates in the dataset. The uncertain algorithm suffers from noticeable increase in the running time because of the additional uncertain decisions, which in turn results in exponential increase in the number of c-records.

Effect of Parameter Range. Figure 9(d) shows the effect of increasing the width of the parameter range that we consider in uncertain deduplication algorithm. Increasing the width of parameter range results in converting more certain decisions into uncertain decisions which increases the number of generated c-records, and hence the running time.

7.3 Performance Results of NN-based Deduplication

Running time of uncertain NN-based deduplication is almost identical to the deterministic version of the algorithm (Figure 10). This is due to two facts: (1) the number of cleaning decisions is the same in both cases, which is equal to the number of compact sets, and (2) most of the execution time is consumed in obtaining the compact sets rather than constructing the `Dependency-Graph`.

Execution of clean instance extraction is less than 0.2 seconds in all cases. We also notice that the running time of α-certain queries is less than 0.1 seconds in all cases. This is due to the simple dependencies among vertices in the `Dependency-Graph`.

Effect of K. Figure 10(b) shows the effect of changing the value of K. We notice that the increase in the running time is insignificant in both uncertain and deterministic cases. The reason is that the number of compact sets of a specific size declines quickly as the size increases. For example, the number of compact sets of size 2 represents 73% of the total number of compact sets.

Effect of Percentage of Duplicates. Figure 10(c) shows the effect of the percentage of duplicates. The NN-based deduplication is slightly affected by the percentage of duplicates in dataset. The reason is that construction of compact sets relies on the relative distances between records rather than the actual number of duplicates in the dataset. Consequently, the number of compact sets (and hence, the number of uncertain decisions) does not increase significantly for greater percentage of duplicates.

Effect of Parameter Range. Figure 10(d) shows that expanding the parameter range does not introduce new compact sets. Thus, the number of uncertain decisions considered by the algorithm, and subsequently the running time, remain fixed.

7.4 Quality of Uncertain Deduplication

We use *recall* and *precision* metrics to measure the quality of the most probable instance. Recall is the fraction of true pairs of duplicate records reported by an algorithm, and precision is the fraction of record pairs an algorithm returns which are truly duplicate [9]. We use the harmonic mean of precision and recall (called F-measure)

to indicate the overall output quality. We compare the most probable clean instance with the instance representing the highest quality within the specified range. We obtain such instance by performing an exhaustive search over the specified range of thresholds using the deterministic algorithm. Note that users cannot be expected to guess the threshold value that generates this best case cleaned instance. We assume that the cleaning parameter is uniformly distributed over the entire range of possible values.

We notice that the difference between quality of the most probable instance and the maximum quality is less than 0.09. The reason is that the most probable instance represents the clean instance that has the highest chance of being produced by any random run of the algorithm, and hence it materialize decisions' outcomes that are produced by the majority of runs. Such characteristics make the most probable instance robust to incorrect decisions produced at single threshold values.

8 Related Work

A number of integrated data cleaning systems have been proposed with different focuses and goals. For example, AJAX [18] is an extensible and flexible 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. Potter's Wheel [24] is an interactive data cleansing system that integrates data transformation and error detection using spreadsheet-like interface. IntelliClean [21] is a rule based approach to data cleaning that focuses on duplicates elimination. The described approaches share a common weakness of inability to handle and capture uncertainty in the process.

Bhattacharya and Getoor have introduced in [7] a new technique to measure the similarity between records based on their relationship with other entities. The proposed technique is based on hierarchical clustering. Although quality of the proposed distance metric is superior to other metrics, determining the distance cut-off threshold remains a source of uncertainty.

The ConQuer system [3] addresses the duplicate elimination problem with an emphasis on providing consistent query answers. The authors assume that clustering of records is deterministically resolved and that uncertainty only emerges from merging clustered records into a representative records.

9 Conclusion

In this paper, we introduced a novel approach to the problem of duplicate detection, treating deduplication procedures as data processing tasks with uncertain outcomes. We introduced a complete lineage-based uncertainty model that compactly encodes the space of possible clean instances. We provided an extended model that captures the behavior of the cleaning process by modeling uncertain cleaning decisions. We show how to use our model to capture uncertainty in two concrete deduplication approaches:

hierarchical deduplication and NN-based deduplication. We also described how to support new query types and update operations on our model.

References

[1] A. M. Abdelbar and S. M. Hedetniemi. Approximating maps for belief networks is np-hard and other theorems. *Artificial Intelligence*, 1998.

[2] C. Andrieu, N. de Freitas, A. Doucet, and M. Jordan. An introduction to mcmc for machine learning, 2003.

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

[4] L. Antova, C. Koch, and D. Olteanu. 10^{10^6} worlds and beyond: Efficient representation and processing of incomplete information. In *ICDE*, 2007.

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

[6] O. Benjelloun, A. D. Sarma, A. Halevy, and J. Widom. ULDBs: databases with uncertainty and lineage. In *VLDB*, 2006.

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

[8] C. Borgelt and R. Kruse. *Graphical Models – Methods for Data Analysis and Mining*. John Wiley and Sons, 2002.

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

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

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

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

[13] Y. C.Yuan. Multiple imputation for missing data: Concepts and new development. *SAS paper*, pages 267–25, 2002.

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

[15] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.*, 19(1), 2007.

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

[17] A. Fuxman and R. J. Miller. First-order query rewriting for inconsistent databases. *J. Comput. Syst. Sci.*, 73(4):610–635, 2007.

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

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

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

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

[22] LinkPlus.

[23] A. McCallum, K. Nigam, and L. H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *KDD*, 2000.

[24] V. Raman and J. M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *VLDB*, 2001.

[25] B. J. Tepping. A model for optimum linkage of records. *Journal of the American Statistical Association*, 1968.

[26] A. Utsugi and T. Kumagai. Bayesian analysis of mixtures of factor analyzers. *Neural Computation*, 2001.