Query Analytics over Probabilistic Databases with Unmerged Duplicates

Ekaterini Ioannou and Minos Garofalakis

Abstract—Recent entity resolution approaches exhibit benefits when addressing the problem through *unmerged duplicates*: instances describing real-world objects are not merged based on apriori thresholds or human intervention, instead relevant resolution information is employed for evaluating resolution decisions during query processing using *"possible worlds"* semantics. In this paper, we present the first known approach for efficiently handling complex analytical queries over probabilistic databases with unmerged duplicates. We propose the ENTITY-JOIN operator that allows expressing complex aggregation and iceberg/top-k queries over joins between tables with unmerged duplicates and other database tables. Our technical content includes a novel indexing structure for efficient access to the entity resolution information and novel techniques for the efficient evaluation of complex probabilistic queries that retrieve analytical and summarized information over a (potentially, huge) collection of possible resolution worlds. Our extensive experimental evaluation verifies the benefits of our approach.

1 Introduction

Entity Resolution is the task of processing a data set in order to create **entities** by merging the data set **instances** that describe the same real-world **objects**. This task is especially important when integrating data from various sources, since typically each source can use different descriptions for the same real-world object. Due to its importance, entity resolution has been widely studied by the database community [15], [23], and the proposed approaches touch various aspects of the problem, including string similarity [8], [11], collective resolution (using relationships between instances) [13], [21], and iterative resolution mechanisms [7].

Modern applications (e.g., Web 2.0) have introduced new challenges to the resolution problem, including higher levels of heterogeneity, and more frequent data modifications [33]. Unfortunately, existing resolution techniques are not able to address these new challenges [15], since they are based on an a-priori merging of instances: they first detect the possible matches between instances and then, given a threshold, decide which instances to merge into entities. The entities resulting from the merges are then used for replacing the coreference instances in the original data set.

To handle these new resolution challenges, recently introduced approaches have moved towards databases that maintain *unmerged duplicates* [4], [19], [31]. These approaches perform only the first part of the resolution process, which is the identification of the possible matches between the instances. Such resolution information has different forms, e.g., clusters of instances that describe the same real-world object [4], [31], or linkages between pairs of matching instances [19]. In some cases the resolution information is accompanied by probabilities, encoding the inherent uncertainty of the resolution process. This information is not used for performing merges between instances (e.g., based on a threshold), but is maintained alongside the original data to capture appropriate *"possible worlds" semantics* at query time [4], [19]. The focus of these approaches is on using the resolution information for processing *simple (single-table) queries*, with probabilistic answers reflecting the different possible real-world scenarios, i.e., resolution decisions.

Although answering simple queries over unmerged duplicates is important, it is still just a first step towards a complete solution to in-database entity resolution. The typical situation is that the unmerged duplicates are part of a large database that, of course, contains other tables. Consequently, users would require retrieving information related to all data, and not only the table with the unmerged duplicates. In addition, queries returning all answers resulting from all possible resolution scenarios can easily overwhelm the user, as the number of these scenarios is huge [12]. In such situations, users typically do not even care about the exact information in individual entities; rather, their main focus is on efficiently obtaining *aggregate, statistical insights about the collection of resulting entities*, similar, in spirit, to online analytical processing.

Our Contributions. In this work, we introduce the first known generic approach for processing complex analytical queries over probabilistic databases with unmerged duplicates. Our solution offers several benefits with respect to existing approaches. First, it adopts the most expressive form of resolution information (i.e., probabilistic linkages between instances – also accounting for transitivity), and significantly extends its scope by considering the resolution information as part of a database with other tables providing entity-related data. Second, it introduces a novel indexing structure that provides efficient query-time access to the resolution information, and the resulting merges and their probabilities. Third, it is the first to enable the efficient execution of complex analytical queries over the unmerged duplicates and their related data.

• *E. Ioannou and M. Garofalakis are with the Technical University of Crete, Chania, Greece. E-mail: {ioannou,minos}@softnet.tuc.gr*

Buyer

id	*name*	*surname*	*loc.*	*gender*	*year*
r_1	Marion	Smith	GR	female	2009
r_2	Marion	Smith	DE	female	2010
r_3	Mary	Smith	DE	female	2011
r_4	John	Smith	GR	male	2010
r_5	Johnny	Smith	GR	male	2011

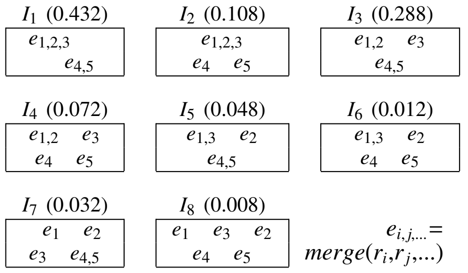
Resolution

id	*instance_1*	*instance_2*	*probability*
l_{r_1,r_2}	r_1	r_2	0.9
l_{r_1,r_3}	r_1	r_3	0.6
l_{r_4,r_5}	r_4	r_5	0.8

Order

id	*buyer*	*items*	*amount*
t_1	r_1	1	20
t_2	r_2	2	150
t_3	r_2	4	300
t_4	r_3	2	40
t_5	r_3	2	60
t_6	r_4	2	30
t_7	r_4	1	10
t_8	r_5	2	40

Fig. 1. A small fragment of a database with unmerged duplicates.



I_1 (0.432)
$e_{1,2,3}$ $e_{4,5}$
I_2 (0.108)
$e_{1,2,3}$ e_4 e_5
I_3 (0.288)
$e_{1,2}$ e_3 $e_{4,5}$
I_4 (0.072)
$e_{1,2}$ e_3 e_4 e_5
I_5 (0.048)
$e_{1,3}$ e_2 $e_{4,5}$
I_6 (0.012)
$e_{1,3}$ e_2 e_4 e_5
I_7 (0.032)
e_1 e_2 e_3 $e_{4,5}$
I_8 (0.008)
e_1 e_3 e_2 e_4 e_5
$e_{i,j,...} = merge(r_i, r_j, ...)$

Fig. 2. Possible worlds and their merges.

id	**name**	**surname**	**loc.**	**gender**	**year**
e_1	Marion	Smith	GR	female	2009
e_2	Marion	Smith	DE	female	2010
e_3	Mary	Smith	DE	female	2011
$e_{1,2}$	Marion	Smith	DE	female	2010
$e_{1,3}$	Mary	Smith	DE	female	2011
$e_{1,2,3}$	Mary	Smith	DE	female	2011
e_4	John	Smith	GR	male	2010
e_5	Johnny	Smith	GR	male	2011
$e_{4,5}$	Johnny	Smith	GR	male	2011

TABLE 1
The entities contained in the possible worlds.

Our approach supports complex aggregation and iceberg/top-k queries. A vital operator in the supported syntax is a novel ENTITY-JOIN operator, which allows expressing joins between tables with unmerged duplicates and other database tables. The ENTITY-JOIN can be used for query analytics using either aggregation operators (e.g., `sum`, `cnt`, `min`, and `max`, in combination with `range`, `variance`, and `mean`) or iceberg/top-k operators. Coupled with our query processing algorithm, these operators allow users to efficiently retrieve statistical information about the possible resolved entities. Each query answer is accompanied by a probability reflecting the possible worlds in which it appears. Our contributions can be summarized as follows:

1. We propose the first-known solution of processing complex probabilistic queries over probabilistic databases with unmerged entity duplicates. Our focus is on practical scenarios that do not require retrieving the huge collection of all possible resolution worlds, but rather analytical or summarized information on the entities.

2. We introduce the ENTITY-JOIN operator for performing joins considering the possible resolved entities. We formally define the semantics of queries with ENTITY-JOIN focusing on two analytical query types: aggregation and iceberg/top-k queries.

3. We present a novel indexing structure and processing algorithms that enable efficient query-time access to the resolution information, their merges, and their probabilities.

4. We provide new techniques that exploit efficient processing aggregation and iceberg/top-k queries without the need to materialize the possible worlds.

5. We validate our techniques through an extensive evaluation (using three real-life data sets), and also report comparison results using alternative methodologies.

2 Motivating Example

Consider an online store that integrates data from a set of similar systems. Figure 1 shows a small fragment of the store's database. Table Order is a deterministic table, whereas table Buyer contains duplicate instances, i.e., instances that may describe the same real-world objects. Executing a resolution algorithm on the Buyer table will generate probabilistic resolution information that represents the possible matches between its instances (table Resolution). For instance, linkage l_{r_1,r_2} states that with probability 0.9, instance r_1 from table Buyer describes the same real-world object as instance r_2. Accepting this linkage implies a new entity, with identifier $e_{1,2}$, that replaces both r_1 and r_2. The system incorporates a merge function for specifying the data representing the final entity. As we discuss later (Section 3), this merge function can have different semantics. For this example, we assume that it returns the instance with the highest value on the year attribute. Therefore, the tuple for the merge between instances r_1 and r_2 is $\langle e_{1,2}$, "Marion", "Smith", "DE", "female", "2010"$\rangle$. Table 1 shows the resulted tuples for all possible merges between instances of the Buyer table.

Combining the data of tables Resolution and Buyer, models different real-world situations, depending on which linkages we actually accept. These are termed *possible worlds* (Figure 2). Accepting all three linkages, for instance, results in possible world I_1 that contains two entities: one created by the merge of r_1, r_2 and r_3, and another by the merge of r_4 and r_5. Typically [4], [19], [31], we assume that linkages are independent, and thus the probability of each world (shown in parentheses in the figure) depends on the probability of the accepted and rejected linkages; for example, the probability of possible world I_1 is equal to the product of the linkages l_{r_1,r_2}, l_{r_1,r_3}, and l_{r_4,r_5}. The idea of tracking unmerged duplicates and possible resolution worlds has been recently introduced in the database community, e.g., in the data models of [4], [19], [31]. In this work, we follow and extend the model of [19], which is the most generic model from the ones suggested since it captures arbitrary probabilistic linkages. In addition, it also accounts for the required transitivity among linkages, i.e., using only the valid of possible worlds.

Consider now that we want to join the entities created by these merges with the data from table Order. This implies a set of records for each entity (e.g., for I_1, t_1-t_5 would correspond to $e_{1,2,3}$), typically yielding a huge (exponentially-large) number of possible result records, considering the huge number of possible resolution worlds. Generating all these scenarios is infeasible, and the huge volume of results would make it impossible for users to derive any meaningful information. To remedy this, we integrate various analytical operators and qualifiers, operating at different levels of aggregation. The **first aggregation level** is *within each possible resolution world*, using conventional SQL aggregate semantics over the merged entities. For instance, summation over the "amount" attribute of table Order for each entity generates:

I_1 : $\langle e_{1,2,3}$, ..., DE, 570$\rangle$, $\langle e_{4,5}$, ..., GR, 80$\rangle$
I_2 : $\langle e_{1,2,3}$, ..., DE, 570$\rangle$, $\langle e_4$, ..., GR, 40$\rangle$, $\langle e_5$, ..., GR, 40$\rangle$
I_3 : $\langle e_{1,2}$, ..., DE, 470$\rangle$, $\langle e_3$, ..., DE, 100$\rangle$, $\langle e_{4,5}$, ..., GR, 80$\rangle$
I_4 : $\langle e_{1,2}$, ..., DE, 470$\rangle$, $\langle e_3$, ..., DE, 100$\rangle$, $\langle e_4$, ..., GR, 40$\rangle$, $\langle e_5$, ..., GR, 40$\rangle$
I_5 : $\langle e_{1,3}$, ..., DE, 120$\rangle$, $\langle e_2$, ..., DE, 450$\rangle$, $\langle e_{4,5}$, ..., GR, 80$\rangle$
I_6 : $\langle e_{1,3}$, ..., DE, 120$\rangle$, $\langle e_2$, ..., DE, 450$\rangle$, $\langle e_4$, ..., GR, 40$\rangle$, $\langle e_5$, ..., GR, 40$\rangle$
I_7 : $\langle e_1$, ..., GR, 20$\rangle$, $\langle e_2$, ..., DE, 450$\rangle$, $\langle e_3$, ..., DE, 100$\rangle$, $\langle e_{4,5}$, ..., GR, 80$\rangle$
I_8 : $\langle e_1$, ..., GR, 20$\rangle$, $\langle e_2$, ..., DE, 450$\rangle$, $\langle e_3$, ..., DE, 100$\rangle$, $\langle e_4$, ..., GR, 40$\rangle$, $\langle e_5$, ..., GR, 40$\rangle$

TABLE 2
Records created by the first aggregation level.

Although the result is a smaller number of records with aggregated information, it is still of exponential size (i.e., linear in the number of possible worlds), and, thus, difficult for users to analyze for reaching vital business decisions. We therefore support a **second aggregation level**, *across all possible resolution worlds*, over all the records created by the first level and based on one (or more) query attributes of interest.

Our probabilistic, second-level aggregation semantics, can express and evaluate queries with different analytical operators over the collection of possible worlds. We now describe a few such examples.

• **Iceberg/Top-k Probabilistic Aggregates:** One useful query type is top-k that allows users to find the *high-probability resolution scenarios* that satisfy specific selection predicates. As an example, consider the following top-k query:

```
SELECT top-2 entity_amount, prob
FROM Order entity-join Buyer
  based on Resolution                                   (Q1)
  using sum(Order.amount) as entity_amount
WHERE Buyer.year=2010
```

This aims to compute the two most likely aggregate amounts spent by buyers in 2010, along with their respective probabilities. (Instead of top-*k*, a query could simply specify a lower bound on the probability of the aggregate values returned.) The entities satisfying the WHERE conditions are e_2 from possible worlds $I_{5,6,7,8}$, $e_{1,2}$ from $I_{3,4}$, and e_4 from $I_{2,4,6,8}$[1]. Their probabilities are the summation of the probabilities of the worlds in which they participate, i.e., 0.1 for e_2, 0.36 for $e_{1,2}$, and 0.2 for e_4. By default, the entities are ordered by probability, thus, the answer is {⟨470, 0.36⟩, ⟨40, 0.2⟩}.

• **Aggregate Ranges:** One basic statistical summary is the range of possible aggregated values over all possible worlds. As an example, let us consider a manager that wants to retrieve the range of possible total Order amounts per location, and thus poses the following query:

1. Notation $I_{i,j,...,k}$ is used to denote possible worlds I_i, I_j, ..., and I_k.

```
1  SELECT Buyer.location, range(entity_amount), prob
2  FROM Order entity-join Buyer
3    based on Resolution                                (Q2)
4    using sum(Order.amount) as entity_amount
5  GROUP BY Buyer.location
```

Although not directly expressed in the query, the ENTITY-JOIN (lines 2-4) implies aggregation of the records corresponding to each entity in the possible worlds by assuming an implicit group-by operator over the entities (aggr. level 1, see Table 2). Evaluating the (explicit, line 5) GROUP BY clause over the resulting records gives two locations: "GR" and "DE" (aggr. level 2). Consider now all entities in the possible worlds. As shown in Table 2, the amount summation for location "GR" is between 20 and 80, which means that the range is [20-80]. For location "DE", it is between 100 and 570, and thus the range is [100-570]. As both locations are present in all possible worlds, their probability is 1. These location-range pairs along with their probabilities compose the answer set for query Q2: {⟨"GR", [20-80], 1⟩, ⟨"DE", [100-570], 1⟩}.

• **Range Drill Down Qualifier:** In the above examples, we showed how to retrieve summarized information. However, users can also be interested in retrieving results with more details, probably after executing aggregation queries, which basically implies reversing parts of the performed summarization. We support this through the *"drill down"* qualifier. As our focus is on entity resolution, we use this qualifier for returning more details with respect to the underlying resolution decisions, i.e., DRILL DOWN includes sub-ranges for each subset of instances that are mutually connected through (possible) linkages. Note that the instances of such a *mergeable subset* could potentially all merge to a single real-world object, while linkage decisions across such subsets are completely independent.

Our example (Figure 1), has two mergeable subsets of instances, namely $\{r_1, r_2, r_3\}$ and $\{r_4, r_5\}$. Reposing our example query Q2 with a DRILL DOWN splits the range of each location into sub-ranges, one for each mergeable subset of instances. For location "GR" and instances $\{r_4, r_5\}$, we have range [40-80] with probability 1 because this is present in all the possible worlds. For location "GR" and entities $\{r_1, r_2, r_3\}$, the range is [20-20] and with probability 0.04 as this is present in I_7 and I_8. Similarly, we can see that for location "DE" and $\{r_1, r_2, r_3\}$ we have range [100-570] with probability 1, and no range for location "DE" and $\{r_4, r_5\}$. Thus, the answer set for query Q3 that includes a DRILL DOWN is: {⟨"GR", [20-20], 0.04⟩, ⟨"GR", [40-80], 1⟩, ⟨"DE", [100-570], 1⟩} (note that DRILL DOWN does not give disjoint alternatives, i.e., ranges might correspond to overlapping possible worlds and thus the sum of range probabilities can be greater that 1). Comparing these results to the ones from the original Q2, the manager understands that there can be no entities resulting in GR order values between 21 and 39.

• **Statistical Operators:** Two additional complementary functions that provide insightful information over possible worlds are: (i) `mean` that shows the average value over the ranges of all possible merges, and (ii) `variance` that indicates the typical discrepancy of the expected value. As an example, consider again query Q2 but with a `mean` and a `variance` aggregate function instead of `range` (line 1). Both are computed based on the mergeable subsets of instances. As shown in the drill down example, for location "GR" we have two ranges: [40-80] for the mergeable subset r_4-r_5, and [20-20] for r_1-r_3. The former range has a mean value of 60 and corresponds to 3 distinct entities[2], whereas the latter has a mean value of 20 and corresponds to 1 distinct entity. Thus, the value of `mean` is [3×60+1×20]/(3+1)=50, and the value of `variance` is $[3\times(60\text{-}50)^2+1\times(20\text{-}50)^2]/(3+1)=300$ (i.e., standard deviation ≈ 17)[3]. As such, the manager realizes that the total amount of orders for location "GR" is typically between 33 and 67 (within one std. deviation of the mean).

Note that the repertoire of aggregate/analytical operators defined here is similar, in spirit, to [31]. However, there exist the following crucial differences: (i) the model followed in [31] assumes that the algorithm is provided with fixed clusters of instances, which allows them to primarily focus on basic query-time aggregation. In sharp contrast to [31], we incorporate a more generic resolution model that requires dealing also with linkages between instances as well as linkage transitivity; (ii) we also consider *probabilistic linkages*, capturing the relevant entity-linkage uncertainty; and, (iii) we support a more expressive query syntax in comparison to [31], which includes two aggregation levels, additional aggregation functions, and iceberg/top-k queries.

Ideally, this problem should be handled by modeling the data using a representative graphical model. However, this would restrict the approach to the most probable situation as oppose to our algorithm that allows retrieving a larger number of possible worlds that interest the users. In addition, executing the inference process (required for retrieving this most probable world) is prohibitively expensive and cannot be applied on data sets of real-world applications.

3 Foundations

In this work, we follow and extend the data model introduced by [19], which involves duplicated instances maintained in an unmerged state. More specifically, our model involves instances that describe the same real-world object, and linkages that express the belief (i.e., through probabilities) that two instances should be merged.

It is important to note that our model is different from conventional tuple-independent probabilistic data models (e.g., [12]) in several important aspects, including the specification of probabilities on *pairs* of instances (i.e., linkages), and the linkage transitivity requirement that, among other things, implies the need for *reasoning at query time over subsets of linkages*. (This distinction is discussed further in Section 9.) Although we focus on the probabilistic linkages model, the ideas we introduce can also be incorporated in existing techniques for probabilistic data, such as [5], [28].

2. Note that entities resulting from merges between different subsets of instances are considered as distinct outcomes.

3. Other definitions can also be accommodated by our techniques, e.g., using the cumulative probability of each range instead of the number of corresponding entities.

3.1 Data Model

We assume the existence of an infinite set of instance identifiers $\mathcal{O}$, names $\mathcal{N}$, and atomic values $\mathcal{V}$. An instance is a design artifact used to model a real-world object. It consists of a unique identifier and a set of attributes. An *attribute* is a pair $\langle n,v\rangle$ of a name and a value, describing some characteristic of the instance. The set $\mathcal{A}=\mathcal{N}\times\mathcal{V}$ represents the infinite set of all the possible attributes.

Definition 1. *An* instance *r is a tuple $\langle id, A\rangle$ where $id\in\mathcal{O}$ is the* identifier *and $A\subseteq\mathcal{A}$ is a finite set of* attributes. ∎

We assume the existence of an infinite set of instances $\mathcal{R}$, and our database contains a subset of these instances, i.e., $R\subseteq\mathcal{R}$. There are instances among R that describe the same real-world objects. Such relationships between the instances of R are encoded by the *linkage set*, defined as a binary symmetric relation $L\subseteq R\times R$. Each element $(r_\alpha,r_\beta)\in L$, also denoted as l_{r_α,r_β}, indicates the belief that r_α is the same real-world object as r_β.

Definition 2. *The factor set $\{\mathfrak{f}_1, ..., \mathfrak{f}_n\}$ is a partition of linkages L into disjoint linkage sets, with each factor being a maximal group of pairwise linked instances: (i) $\forall l_{r,r'}\in\mathfrak{f}_i$, if $l_{r,r''}\in L \Rightarrow l_{r,r''}\in\mathfrak{f}_i$ (ii) $\mathfrak{f}_i\cap\mathfrak{f}_j=\emptyset$, and (iii) $\bigcup_{i=1..n}\mathfrak{f}_i=L$.*

The mergeable subset of factor $\mathfrak{f}_j$ is the distinct union of the instances participating in the linkages of $\mathfrak{f}_j$, i.e., $\bigcup\{r,r'\}$ $\forall l_{r,r'}\in\mathfrak{f}_j$ and provides instances that could be merged into entities. ∎

Factors are essentially connected components of the linkage graph. The mergeable subset of each factor, i.e., the instances contained in the linkages of the specific factor, provide instances that could potentially merge into one or more real-world objects. On the contrary, the instances contained in the mergeable subset of different factors can never participate in the same merge.

Example 1. *Consider again the five instances from the example shown in Figure 1, i.e., l_{r_1,r_2}, l_{r_1,r_3}, l_{r_4,r_5}. We need to place each instance in a factor. The first linkage contains an instance that is also found in the second linkage, i.e., instance r_1 participates in linkages l_{r_1,r_2} and l_{r_1,r_3}. We thus place the instances from these linkages in the same factor. The instances of the third linkage participate only in the specific linkage, thus we create a new factor that contains these instances. This results in $\mathfrak{f}_1=\{l_{r_1,r_2}, l_{r_1,r_3}\}$ and $\mathfrak{f}_2=\{l_{r_4,r_5}\}$ and thus the mergeable subsets of instances $\{r_1, r_2, r_3\}$ and $\{r_4, r_5\}$.*

Having these linkages separated into two factors means that we only need to consider creating entities by merging instances participating in the linkages of $\mathfrak{f}_1$ or $\mathfrak{f}_2$. For example, we do not have to consider merging instances r_1 with r_4 since these instances participate in linkages from different factors, i.e., r_1 in linkages of $\mathfrak{f}_1$ and r_4 in linkages of $\mathfrak{f}_2$. ∎

Partitioning linkages into factors has positive effects on efficiency, as we now need to handle linkage subsets of smaller sizes. Clearly, the factor characteristics (e.g.,

```
1  SELECT TOP-K column_j, ..., column_k, prob |
2                  grp_column, aggFunct(agg_column), prob
3  FROM  T_i entity-join T_dup based on T_res
4        using emAggFunct(T_i.column) as agg_column
5        [having ent_probability ≥ double]
6  [WHERE  conditions]
7  [GROUP BY  grp_column [with drill-down] ]
8  [HAVING  grp_probability ≥ double]
                        emAggFunct: min | max | sum | cnt
                         aggFunct: range | mean | variance
```

Fig. 3. The syntax of an ENTITY-JOIN query.

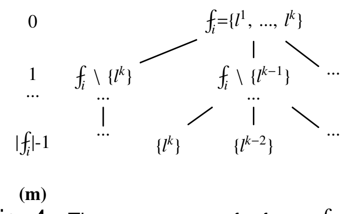


Fig. 4. The tree structure for factor f_i.

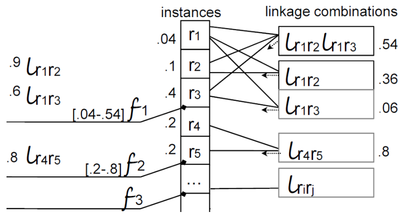


Fig. 5. Indexing structure for Figure 1.

number of linkages) depend on the resolution technique used for generating the linkage set. In essence, we cannot make any assumption with respect to these characteristics since the linkage set is given as an input to our approach. Our goal is to introduce a generic methodology that is able to operate efficiently over different linkage characteristics. The influence of characteristics on performance is further explored in Section 8.

DEFINITION 3. *The entity $e_{1,...,n}$, created by function merge(r_1, ..., r_n), represents the set of pairwise linked instances r_1, ..., r_n.* ∎

We stress that the merge function, as included in the above definition, is generic enough to capture the different possible semantics of merging instances. One possibility is *merge* | $\{r_1, ..., r_n\} \mapsto r_i$ with $i \in [1,n]$, which is followed by methods such as [9], considering that one of the instances will be used for representing the final merge. Another possibility for the merge function is *merge* | $\{r_1, ..., r_n\} \mapsto r'$ with $r'.A \subseteq \cup r_i.A$ and $i \in [1,n]$. This merge function creates the final entity by composing attributes from some/all of the instances to be merged, as for example performed in [35]. In the current version of our work, we use the first function, i.e., one of the instances is used for representing the final merge. However, incorporating a merge function of the second type does not alter the techniques introduced for query evaluation (Section 7). In what follows, we will also use $e_{1,...,n}$ as the identifier of the entity returned by *merge*(r_1, ..., r_n), and since entities are the results of merges, the terms entity and merge will be used interchangeably.

We consider a database that maintains the data of a complex information system. As such, our database does not only contain information related to instances and their resolution, but it also contains other tables with additional information. These tables can, and typically will, also have references to the instances in R.

DEFINITION 4. *A database D with unmerged duplicates is a tuple $\langle T_1, ..., T_n, R, L, p^l\rangle$, where T_1, ..., T_n are deterministic relational tables, R is a table containing duplicates, L contains linkages over the instances in R, and p^l is the linkage probability assignment function $p^l \mid L \mapsto [0, 1]$.* ∎

Note that, compared to the data model of our earlier work [19], the key difference here is that we consider the scenario in which the duplicated instances are part of a database, i.e., with other tables, over which we need to provide efficient processing of complex queries. Also note that the above definition can be extended to include more than one tables with duplicate instances (Section 7).

An uncertain database with unmerged duplicates (Definition 4) contains the linkage set L and the linkage probability assignment function p^l that assigns probabilities to the L linkages. This means that each linkage exists with the probability given by p^l and does not exist with probability 1-p^l. We can therefore create various situations (*possible worlds*), by deciding which linkages from L to accept or not (termed *linkage combinations*). The combinations for each world are used for defining the entity merges (e.g., Figure 2).

An ENTITY-JOIN is the probabilistic join between table T_i with each of the entities in the possible worlds. Note that a possible world will be considered in the join only when this is valid, i.e., the transitivity of the accepted linkages does not violate the transitivity of rejected linkages. This means that if the accepted linkages imply entity $e_{...,\alpha...,\beta,...}$, then the rejected linkages must also indicate that instance r_α can be merged with instance r_β.

3.2 Analytical Queries over Unmerged Duplicates

Figure 3 shows the complete query syntax that can be processed given an uncertain database with unmerged duplicates, i.e., $\langle T_1, ..., T_n, R, L, p^l\rangle$. The supported query syntax is given as an extension of SQL syntax, in order to illustrate our approach's capability for easy usage by users and simple integration with traditional relation databases.

The query semantics focus on enabling users to retrieve analytical or summarized information about the entities and related uncertainties. The basis for this task is the set of possible resolved entities that can be created given the duplicated instances T_{dup} and the probabilistic linkage information T_{res}. The FROM clause, and more specifically the ENTITY-JOIN operator (Figure 3, lines 3-4), states that the possible resolved entities must be joined with the records of table T_i. Here, we assume a group by operator over each possible merge (i.e., entity). Thus, the corresponding result set must include one record per entity. Although this group by operator is not directly included in the query syntax, it is implicitly expressed through the `emAggFunct` aggregation function (line 5). The supported aggregation functions are `min`, `max`, `sum`, and `cnt`.

Example 2. *Consider again the* ENTITY-JOIN *from query Q1. As explained, our query semantics assume a group by operator over the records of each entity. For example, having entity $e_{4,5}$ from I_1 gives the following records as the results of joining Order and Buyer:*
- *⟨$e_{4,5}$, "Johnny", "Smith", "GR", "male", "2011", 2, 30⟩*
- *⟨$e_{4,5}$, "Johnny", "Smith", "GR", "male", "2011", 1, 10⟩*
- *⟨$e_{4,5}$, "Johnny", "Smith", "GR", "male", "2011", 2, 40⟩*

Executing the `sum` *aggregation function over these records,* *as required by line 4 in Q1, creates the following single record for the specific entity:*

• *⟨$e_{4,5}$, "Johnny", "Smith", "GR", "male", "2011", 5, 80⟩*

Let us now assume that the requested aggregation function is a `cnt`*. Then, the resulted record for the $e_{4,5}$ entity is:*

• *⟨$e_{4,5}$, "Johnny", "Smith", "GR", "male", "2011", 3, 3⟩*

∎

Over the entities created by the **FROM** clause we can execute aggregation and iceberg/top-k. Aggregation queries allow users to include an additional aggregation level over the resulting entities across all possible worlds. This query type is expressed by specifying a **GROUP BY** clause (line 7) and an accompanying `aggFunct` (line 2). Iceberg and top-k queries allow users to derive high-probability distinct entities across all possible worlds (line 1 & 5).

In the following paragraphs we further explain the semantics of these query types.

I. Aggregation Queries. The current version of our approach can execute three aggregation functions over the entities resulted from the ENTITY-JOIN (line 2). One of these aggregation function is `range`. For this function, the entities of all possible worlds are grouped by grp_column (line 7), and the function returns the lower and the upper values among the grouped data.

Aggregation queries with a `range` function are equipped with two optional qualifiers. The first is HAVING qualifier (line 8) that allows setting a lower probability threshold for filtering the merges. The second qualifier is DRILL-DOWN (line 8) that returns sub-ranges (instead of just one range) for each agg_column value. These sub-ranges are derived by combining an agg_column value with each one of the mergeable instance subsets. Thus, each returned sub-range represents the lower and upper values for instances with common linkages. A query example that includes a DRILL-DOWN qualifier was discussed in Section 2.

The other two aggregation functions are `mean` and `variance`. Both functions perform the computation by combining a grp_column value (line 7) with each one of the mergeable instance subsets (similar to the DRILL-DOWN qualifier). They then compute `mean`/`variance` of each grp_column value. For `mean`, this is the summation of every range midpoint multiplied with the number of entities in this range, and divided with the number of entities in all ranges. The computation for `variance` is similar, but is based on the squared numerical difference of the range midpoint with the computed mean value. Both functions require knowing the total number of entities for each range, which can be hard to compute; thus, we instead use a sampling-based estimate (explained in Section 5). Note that a query example with a DRILL-DOWN qualifier was discussed in Section 2.

II. Iceberg & Top-k Queries. In addition to executing aggregate queries over the entities created by the ENTITY-JOIN operator, users can also filter the entities using iceberg or top-k queries. Both query types consider distinct entities across all possible worlds, where the probability of each entity is accumulated across all the possible worlds in which it is present.

Iceberg queries allow setting a lower limit on the probability of the entities. This probability is given by including a HAVING qualifier (line 6) in the query. Top-k queries state that only k entities should be returned, the ones with the highest aggregate probabilities (i.e., across all worlds). For defining such a query, users need to include a TOP-K qualifier (line 1). An example of a top-k query was presented in the last paragraphs of Section 2.

4 Indexing Structure

We now introduce the indexing structure, which forms the basis of the efficient processing of the supported query types. The main goal of the indexing structure is to reduce the complexity of the computations required when processing queries. This is achieved since the indexing structure provides efficient access to the information encoded through the linkages (i.e., potential merges) and since it allows easy construction of possible worlds (or, parts thereof) as well as the fast retrieval of their probabilities.

4.1 Construction

Algorithm 1 illustrates the construction of our indexing structure. The first part creates the factors by detecting connected components in the given linkage collection L (lines 2-8). Each linkage is placed in a factor, such that the instances of this linkage are only referred to by linkages of the same factor. This step's complexity is O($|L|$).

Once we have created the factors for the given linkage collection (lines 2-8), we processes the linkages in each factor (lines 9-18). This involves processing each factor for computing the probability range of all the possible worlds that can be generated using the linkages of the specific factor (line 12), creating a list with the instances participating in the linkages of the factor by taking the distinct union of the linkage instances (lines 13-14), and loading the linkage combinations that can be generated given this factor (lines 15-17). We elaborate on these steps in the following paragraphs.

One of the steps involves computing the ranges for the probabilities of all possible worlds for each factor (line 12). We will use notation $[\underline{v}\mathrm{f}_i^{pr}, \overline{v}\mathrm{f}_i^{pr}]$ to denote that the probability range for factor f_i has a lower value $\underline{v}\mathrm{f}_i^{pr}$ and an upper value $\overline{v}\mathrm{f}_i^{pr}$. To simplify the discussion, and without loss of generality, we now assume that linkage probabilities are all above 0.5. (To handle linkages with probabilities below 0.5, we simply assume that these linkages do not exist and use their complement (see proof of Theorem 1).) We now need to use the resolution information that has the highest probability. Recall that each factor f_i is a set of linkages. Thus, in case the probability p of a linkage is lower than 0.5, we consider the negation of the linkage, i.e., that the two instances do not match, and this occurs with probability $1-p$. If none of the linkages from factor f_i are accepted then each instance will create a unique entity. On the contrary, if all linkages are accepted then we will have one single entity by merging all instances.

Algorithm 1: Construction of the Indexing Structure

```
   Input:  Instances $R$, Linkages $L$, Linkage prob. function $p^l$
   Output: Indexing structure $I$
1  $I \leftarrow \emptyset$;  $F \leftarrow \emptyset$;
2  foreach $l \in L$ do // create factors
3      $f_1 = F$.getFactorWithInstanceInLinkage($l$.instance1);
4      $f_2 = F$.getFactorWithInstanceInLinkage($l$.instance2);
5      if ($f_1$=$f_2$=−1) then  $F = F \cup \{\{l\}\}$;
6      else if ($f_1$=−1 & $f_2$!=−1) then  $F[f_2] = F[f_2] \cup \{l\}$;
7      else if ($f_2$=−1 & $f_1$!=−1) then  $F[f_1] = F[f_1] \cup \{l\}$;
8      else  $F[f_1] = F[f_1] \cup F[f_2]$; $F = F \setminus F[f_2]$;
9  foreach $L \in F$ do // process linkages of each factor
10     elem $\leftarrow \emptyset$;
11     elem.linkages $\leftarrow L$;
12     elem.probability_range $\leftarrow$ computeRange( $L$ ); // Eq. 1
13     foreach $l \in L$ do
14         elem.factor = elem.factor $\bigcup_{distinct}$ {$l$.instance1,
           $l$.instance2};
15     TopK $\leftarrow$ retrieveTopKCombinations( $L$, K ); // Alg. 2
16     foreach $CL \in TopK$ do
17         elem.combinations.addCombinat&References( CL );
18     $I \leftarrow I \bigcup$ {elem};
19 return $I$;
```

The probabilities of these two situations provide the bounds of the probabilities for all possible worlds that can be generated by the linkages of this factor. Therefore, the upper probability bound is given by the product of all linkage probabilities, and the lower probability bound is given by the product of the complements of the linkage probabilities. The range $[\underline{v}f_i^{pr}, \overline{v}f_i^{pr}]$ for factor $\mathfrak{f}_i$ is thus:

$$\underline{v}f_i^{pr} = \prod_{l_{r_i,r_j} \in \mathfrak{f}_i} [1 - p^l(l_{r_i,r_j})], \quad \text{and} \quad \overline{v}f_i^{pr} = \prod_{l_{r_i,r_j} \in \mathfrak{f}_i} [p^l(l_{r_i,r_j})] \qquad (1)$$

The algorithm (lines 15-17) also loads a subset of the linkage combinations from each factor in the index (this algorithm is presented in the following paragraphs). The number of linkage combinations that can be generated depends on the number of linkages in $\mathfrak{f}_i$. More specifically, $\mathfrak{f}_i$ can generate at most $2^{|\mathfrak{f}_i|}$ combinations, where $|\mathfrak{f}_i|$ denotes the number of linkages in $\mathfrak{f}_i$. (This number is only an upper bound, since the transitivity of linkages makes some of the combinations invalid.) Clearly, the number of linkage combinations increases exponentially with the factor size. Our indexing structure aims to provide efficient access to the combinations with the highest probabilities. As discussed, the collection of indexed combinations can be dynamically expanded (during query processing) at low overhead.

The probability of a linkage combination CL reflects the fact that only the specific linkages of factor $\mathfrak{f}_i$ exist (and the remaining linkages do not exist) in a possible world. This corresponds to a possible world in which all instances not participating in the linkages of CL are independent (i.e., not part of any merges). The remaining instances create one entity merge, or even multiple entity merges in case the specific linkages can be separated into independent sets of pairwise linked instances. A possible world is invalid when the transitivity of the linkages from CL is violated by the absence of linkages from $\mathfrak{f}_i \backslash CL$, as also performed in [19]. The probability of a possible world given CL is:

$$P(CL, f_i) = \begin{cases} 0, & \text{if merge from CL is invalid} \\ \prod_{l_{r_i,r_j} \in\ CL} p^l(l_{r_i,r_j}) \cdot \prod_{l_{r_i,r_j} \in\ (f_i \backslash CL)} [1 - p^l(l_{r_i,r_j})] \end{cases} \qquad (2)$$

Example 3. *Consider now having factor $\mathfrak{f}_1 = \{l_{r_1,r_2}, l_{r_2,r_3}, l_{r_1,r_3}\}$. The linkage combination $CL = \{l_{r_1,r_2}, l_{r_2,r_3}\}$, and especially the transitivity between linkages, implies that there is a merge between instances r_1, r_2, and r_3. However, $\mathfrak{f}_1 \backslash CL$ equals to $\{l_{r_1,r_3}\}$ and this states that instances r_1 and r_3 must not be merged. This contradicts the merges generated by CL and for this reason we consider the specific merges as invalid.* ∎

Once a linkage combination is generated, we include in our index the probability of the combination along with its linkages, or linkage sets in case the specific linkages can be separated into factors. In addition, for each set of linkages we create instance references, i.e., references to instances r_i and r_j for each linkage l_{r_i,r_j} in the combination. Recall that for every entity merge we apply the merge function, which returns the instance that will represent the specific merge. We encode this information in the index by including an additional reference from the set of linkages to this entity. Such references are called *merge-representation edges*.

In addition to linkage combinations and their probabilities, our index also maintains all instances. When an instance r_i does not participate in any of the accepted linkages, we must consider an entity that basically corresponds to r_i (i.e., no accepted linkages means that r_i is not merged with other instances).

Retrieval of Linkage Combinations. An important technical problem is how to create the possible worlds in order of decreasing probability. Algorithm 2 describes the process of retrieving the K linkage combinations with the highest probabilities for a specific factor. It incrementally generates such linkage combinations in order of probability and, thus, it is also directly applicable for efficiently retrieving all combinations with probability higher than a given threshold. Furthermore, simple adaptations of Algorithm 2 play a critical role in our aggregate query processing strategies (Section 5).

Our algorithm is based on a tree structure (Figure 4), with each node representing a linkage combination. The root of the tree (lines 2, 3 in Algorithm 2) corresponds to the combination with the highest probability, i.e., $\overline{v}f_i^{pr}$. As explained above, this is the combination in which all linkages exist. Each level of the tree contains combinations that exclude one more linkage compared to the combinations contained in the previous level. We use m to denote the number of linkages excluded from $\mathfrak{f}_i$ to create the combination. The following theorem demonstrates that the maximum probability among the combinations in which m linkages do not exist is higher than the maximum probability we will receive if we increase the value of m.

Theorem 1. *Let $L_{f_i,m}$ denote the m linkages from factor $\mathfrak{f}_i$ that have the lowest probabilities, and CL_m denote the* *linkage combination in which the $L_{f_i,m}$ linkages do not exist, i.e., $CL_m = \mathfrak{f}_i \backslash L_{f_i,m}$. Then $P(CL_m, \mathfrak{f}_i) \geq P(CL_{m+1}, \mathfrak{f}_i)$, where $m \geq 0$ and $m < |\mathfrak{f}_i|$.* ∎ *(Proof in the appendix)*

Given the above theorem, we initially set the value of m to zero (i.e., combination represented by the root) and increase it as processing continues. For example, the second level of the tree contains combinations with one excluded linkage, and the last level with $|\mathfrak{f}_i|$-1 excluded linkages. Thus, the algorithm generates the first child node CL_{m+1} of node CL_m by excluding the linkage with the lowest probability, i.e., $CL_{m+1} = CL_m \backslash \{l^k\}$ where $p^l(l^k) \leq p^l(l^j)$ $\forall$ l^k, $l^j \in CL_m$. We can generate various combinations in which m linkages do not exist. However, we need to retrieve combinations sorted by their probabilities, which is achieved through the following theorem.

Theorem 2. *Let CL_{m+1} denote the linkage combination created by excluding linkage l^j from CL_m, i.e., $CL_{m+1} = CL_m \backslash \{l^j\}$, and CL'_{m+1} is created by excluding l^{j-1}, i.e., $CL'_{m+1} = CL_m \backslash \{l^{j-1}\}$. If $p^l(l^j) \leq p^l(l^{j-1})$, then $P(CL_{m+1}, \mathfrak{f}_i) \geq P(CL'_{m+1}, \mathfrak{f}_i)$.* ∎ *(Proof in the appendix)*

That is, for each node we assume that the linkages of CL_m are sorted by probability, i.e., $CL_m = \{l^1, l^2, \ldots\}$ with $p^l(l^i) \leq p^l(l^{i-1})$. The algorithm follows the above theorem and for each node representing combination $CL_{m+1} = CL_m \backslash \{l^j\}$ generates its right sibling node by excluding l^{j-1} from CL_m ($j \geq 2$).

In short, each child will have probability less than or equal to the probability of its parent, and the right sibling of the node will have probability less than or equal to the probability of its left sibling. Thus, retrieving the top K combinations corresponds to a depth-first expansion of the nodes. However, it could happen that the probability of a node becomes lower than that of a previously visited node. When this happens we should continue the processing from the node with the highest probability. For this, each time we process a node (Algorithm 2), we also generate its right sibling and its leftmost child (lines 12-13). These nodes are included in a list sorted by probability (line 14). The processing always continues from the highest-probability node in the list.

Our algorithm terminates when we process K unique combinations from the priority list (lines 6,9), or all nodes are visited. It is easy to see that complexity is $O(K \cdot \log K)$. The combinations included in our index structure are $K \cdot n$, where n is the number of factors created by the linkages L. The main advantage of this mechanism is that it ignores combinations that are unlikely to be needed for answering queries (due to their low probability). When necessary, query processing techniques can, of course, also generate combinations that are not included in the structure on-the-fly (Section 6).

Example 4. *Figure 5 shows the structure for the data of our motivation example. As shown, instances r_1-r_5 are separated into two factors: factor $\mathfrak{f}_1$ with r_1-r_3 and $\mathfrak{f}_2$ with r_4-r_5. For each factor we compute the probability range given its linkages. For example, the range for factor $\mathfrak{f}_1$ has a lower value $\underline{v}f_1^{pr}$=(1-0.9)×(1-0.6)=0.04 and an upper value $\overline{v}f_1^{pr}$=(0.9)×(0.6)=0.54. In addition, we also maintain probabilities for linkage combinations. For example, instance r_3 is connected to combinations $\{l_{r_1,r_2}, l_{r_1,r_3}\}$ and $\{l_{r_1,r_3}\}$. The former creates entity $e_{1,2,3}$ and it has probability equal to 0.9×0.6=0.54. The latter creates entity $e_{1,3}$ and it has probability (1-0.9)×0.6=0.06. All possible combinations are included in the index, however if we had set a limit of 2 combinations per factor, then only the ones with probabilities 0.54 and 0.36 would be included.* ∎

Algorithm 2: Retrieval of Factor's Linkage Combinations

```
Input: Number of combinations K,   Factor Linkages L
Output: Linkage combinations MaxHeap
1  MaxHeap ← {}; // the top K linkage combinations
2  Node ← createNode(L); // all linkages in first node
3  C ← {Node}; // pending nodes sorted by probability
4  while ( MaxHeap.size()<K & C.size()!=0 ) do
5      process();
6  return MaxHeap;
7  function process() // i.e., node with highest prob
8  begin
9      Node = C[0];   C = C \ C[0];
10     if ( MaxHeap.notContains(Node) ) then
11         MaxHeap ← MaxHeap ∪ Node;
12     sibling_node = Node.createRightSibling();
13     child_node = Node.createLeftmostChild();
14     C ← C ∪_sort {sibling_node, child_node};
```

Note that the summation of the probabilities from all nodes (i.e., representing distinct combinations) of the complete indexing structure is one. However, this is typically not reflected in the final query answer set since some of the represented combinations are invalid and thus ignored (excluded from the answer set).

4.2 Basic Operations

Retrieving groups. Aggregation queries specify a GROUP BY clause. We first detect the instances satisfying the query conditions. These instances are then separated into groups G_1, G_2, ..., such that the instances of each G_j correspond to a different value of the field specified by the GROUP BY clause, i.e., $G_j = \{r_i\}$. For example, processing Q2 requires detecting the instances per location, and thus we get group $G_1 = \{r_2, r_3\}$ for "DE", and $G_2 = \{r_1, r_4, r_5\}$ for "GR".

Retrieving factors. The indexing structure allows us to retrieve the linkages grouped based on the factors in which the linkage instances participate. For the instances of group G_j, this will result in linkage sets S_{j1}, ..., S_{jn}, where $S_{ji} = \{ l_{r_\alpha,r_\beta} \mid r_\alpha \text{ or } r_\beta \in G_j \;\&\; l_{r_\alpha,r_\beta} \in \mathfrak{f}_i \}$. For instance, $G_2 = \{r_1, r_4, r_5\}$ gives factors $S_{21} = \{l_{r_1,r_2}, l_{r_1,r_3}\}$ and $S_{22} = \{l_{r_4,r_5}\}$.

Retrieving the result of a merge. Another operation is retrieving the entity resulting from a merge function. For instance, given a linkage set S_{ji} (created by the process described above), we want to retrieve the entity resulting from the merge of all instances that refer to linkages participating in S_{ji}. For this, we use the structure to first detect the linkage combination that contains all linkages from S_{ji}, and then follow the merge-representation edge to identify the instance that will be returned for the corresponding merge. In addition, we compute the merge's probability using Eq. 2 and $CL{=}S_{ji}$. (Merges with zero probability are, of course, ignored due to transitivity violations.) For example, instance r_3 will be the result for the merge of the linkage set $S_{21}{=}\{l_{r_1,r_2}, l_{r_1,r_3}\}$.

5 Aggregation Queries

The result set for an aggregation query contains tuples $\langle g, v^a, p\rangle$, where g is a possible value for the given GROUP BY attribute, v^a is the computed result for the specific group value g, and p the overall probability for the specific g and v^a. Symbol a denotes the aggregate function that was requested in the ENTITY-JOIN, with `sum`, `cnt`, `min`, and `max`, representing summation, count, minimum, and maximum, respectively. For `mean` and `variance` functions, v^a is a real number, denoting the computed value. For `range`, v^a is range $[\underline{v}^a, \overline{v}^a]$, where $\underline{v}^a/\overline{v}^a$ denotes the lower/upper value.

The evaluation of an aggregation query with an ENTITY-JOIN is performed following a sequence of steps. The **first step** retrieves the instances satisfying the query conditions separated in groups G_1, ..., G_n according to the GROUP BY clause (Section 4.2). The remaining steps further process the instances in each group G_j. For summation and count aggregates, we need to consider all the database records referring to each instance. On the contrary, for minimum and maximum aggregates, we only need to consider one record for each instance and not all the records. We follow this distinction in the description of these methods. We first introduce the techniques for evaluating `range` with summation or count (Section 5.1), followed by the techniques for evaluating `range` with minimum or maximum (Section 5.2). We then extend these techniques for evaluating `mean` and `variance` as well as additional query options (Section 5.3), and finally, we explain the probability computation (Section 5.4).

5.1 Range with Cnt and Sum Functions

As mentioned above, the aggregate functions for count and summation require knowledge of all records that are related to the entities. In addition, their result is monotonically increasing with respect to the number of instances in the merge, since more instances increases the join result size and summation (assuming positive summands). We use these aspects to efficiently compute the range's values.

Range's Lower Value. Given this monotonicity we know that the range's lower value, i.e., $\underline{v}^{sum}$ and $\underline{v}^{cnt}$, appears when the entity is generated using only one instance. Therefore, to compute this value we need to consider only the instances (without linkages). Given the groups G_1, ..., G_n that are returned by the first step, we evaluate the aggregate function on the individual instances of each group G_j. The lowest value among all the values returned for each group is then $\underline{v}^a$ for the specific group. The complexity for this step is O($|RG|$), where RG contains the instances of all G_is.

Range's Upper Value. Computing the upper value is more challenging, as we should also follow the linkages and their combinations. We first retrieve S_{j1}, ..., S_{jk} based on the instances of each group G_j (Section 4.2). We then identify the upper value for each S_{ji}, which we denote with symbol $\overline{v}s^a_{ji}$. Thus, the final value $\overline{v}^a_j$ for group G_j is the maximum of all upper values, i.e., $\overline{v}^a_j = max(\overline{v}s^a_{j1}, ..., \overline{v}s^a_{jk})$. The process for computing $\overline{v}s^a_{ji}$ is performed using two phases, explained in the following paragraphs.

Phase 1: The first phase creates a merge out of all the instances participating in the S_{ji} linkages, and then ensures that this merge is an answer to the given query, i.e., will satisfy the query conditions and the value corresponding to group G_j. For this, we test the merge by retrieving the entity for the specific merge (that basically is an instance, cf. Section 4.2), and check if this results in an instance not in G_j. If this is the case, we remove the instance that causes the violation.

Phase 2: The instances for this merge were retrieved by ignoring linkage transitivity, so we now need to verify if a merge with these instances, or a subset of them, can be created. This is based on an adaptation of the algorithm used for retrieving a factor's combinations (Algorithm 2). Each node is now a merge between a set of instances, and the following levels have one instance less (the instance with the lowest result for the requested aggregate function, i.e., `sum` or `cnt`). A node is considered valid when it can be created using the linkages of S_{ji} that refer to the instances composing the specific merge and does not have transitivity violations (Section 4.2). The value returned by the aggregation function over the first accepted node corresponds to $\overline{v}s^a_{ji}$.

In short, computing the range's upper value needs to process the k S_{ji} corresponding to the specific G_j, and for each S_{ji} it creates a collection of merges by removing one of the S_{ji} linkages in every step. Assuming that S_{ji} contains all the possible linkages (worst case), i.e., L, then the complexity for this process is O($k{\cdot}|$L$|$).

Example 5. *Processing query Q2 gives group $G_1{=}\{r_2,r_3\}$ for "DE", and $G_2{=}\{r_1,r_4,r_5\}$ for "GR". Let us now compute the range for G_2. Executing the aggregate function returns 20 for r_1, and 40 for r_4 and r_5. Thus, $\underline{v}^{sum}_2$ is set to 20. To compute the upper value we first retrieve linkages per factor: $S_{21}{=}\{l_{r_1,r_2}, l_{r_1,r_3}\}$ and $S_{22}{=}\{l_{r_4,r_5}\}$. For the former, we create merge(r_1,r_2,r_3). Since this merge returns entity r_3 that is not part of G_2, we replace it with merge(r_1,r_2). This merge returns entity r_2 that is also not part of G_2, and we thus replace it with merge(r_1). Since this merge is also accepted by the second phase, $\overline{v}^{sum}_2$ is set to 20. For S_{22}, we create merge(r_4,r_5) and since this is accepted by both phases $\overline{v}^{sum}_2$ is set to 80. Thus, $\overline{v}^{sum}$ is equal to max(20,80), and the range for "GR" becomes [20,80].* ∎

5.2 Range with Min and Max Functions

All merges that can be created from the data of a factor will have ranges that are subsets of the factor's range. We start by computing the factor ranges, and then, if needed, continue with the possible merges (through their linkages). Alg. 3 provides an overview.

Step 1 - Factor's ranges: The first step computes the range of the aggregation values for the factors. For factor f_i, this process will result in range $[\underline{v}f^{max}_i, \overline{v}f^{max}_i]$

Algorithm 3: Evaluating min/max aggregate function

```
Input: Group $G_j$={$r_1$, ..., $r_n$}, Aggr. Funct. $a$, Indexing str. $I$
Output: Range $v$, where $v$ is [$\underline{v}^a$, $\overline{v}^a$]
$S \leftarrow I$.getFactors($G_j$);
foreach $S_{ji} \in S$ do
    $f\_r \leftarrow$ getFactorRange($a$, $S_{ji}$.factor_id); // step 1
    // step 2
    $r\_r \leftarrow$ getInstanceRange($a$, $S_{ji}$.distinctInstances());
    if $f\_r.l < r\_r.l$ then $r\_r$.l.limit=$f\_r$.l;
    if $f\_r.u > r\_r.u$ then $r\_r$.u.limit=$f\_r$.u;
    $v \leftarrow v \cup r\_r$;
// step 3
if $r\_r.l.limit$ =-1 then compute($r\_r$.l);
if $r\_r.u.limit$ =-1 then compute($r\_r$.u);
return $v$;
```

for the maximum function, and range [$\underline{v}f_i^{min}$, $\overline{v}f_i^{min}$] for the minimum function. These are the ranges of all the possible values that can be generated by applying the specific aggregation function (i.e., line 5, Figure 3) on any possible world that can be created from the linkages included in the specific factor.

Computing the factor's range is based solely on the instances. More specifically, the range can be retrieved by evaluating the same aggregate function (min or max) on the instances from the factor without considering the linkages. The smaller value among the ones returned from the query is then $\underline{v}f_i^{min}/\underline{v}f_i^{max}$ and the higher value is $\overline{v}f_i^{min}/\overline{v}f_i^{max}$.

Step 2 - Initial Range Values: Once we have computed the ranges for the factors, we retrieve the groups G_1, ..., G_j and their linkages separated into factors S_{j1}, ..., S_{jk} (Section 4.2). We then compute the range for the instances participating in the linkages of each S_{ji} by analyzing the information in the instances, and then, if needed, we also follow the linkages and their combinations. This range is computed similarly to retrieving the range for a factor (explained above), but now we only consider the instances participating in the linkages of S_{ji}. The result is range [$\underline{v}s_{ji}^a$, $\overline{v}s_{ji}^a$], where a denotes the requested aggregate function.

Consider again the range [$\underline{v}f_i^a$, $\overline{v}f_i^a$] for factor $\mathfrak{f}_i$. The resulting range for S_{ji} is given by range [$\underline{v}s_{ji}^a$, $\overline{v}s_{ji}^a$]. We know that $\underline{v}f_i^a$ is actually the lowest possible value of factor $\mathfrak{f}_i$. Thus, if $\underline{v}s_{ji}^a$ is higher than $\underline{v}f_i^a$, then it could be that following some of the related linkages, which we so far ignored, could give us the exact value. We therefore know that the lower value of the range is between $\underline{v}s_{ji}^a$ and $\underline{v}f_i^a$, and the upper value of the range between $\overline{v}s_{ji}^a$ and $\overline{v}f_i^a$.

Step 3 - Detecting Final Range: The challenge for computing the exact range is that we need to also consider instances outside G_j. We first use the index to retrieve the linkages per factor that the instances of G_j refer to, i.e., S_{j1}, ..., S_{jk}. For each S_{ji}, we retrieve the range values for the corresponding factor, i.e., $\mathfrak{f}_i$, and the range values for the instances participating in the linkages of S_{ji}. When $\underline{v}f_i^a$ is less than $\underline{v}s_{ji}^a$ or $\overline{v}f_i^a$ is more than $\overline{v}s_{ji}^a$ we need to search for the value. As we want to compute the range over all data in group G_j, the following step performs the union of all S_{ji} ranges.

We might need to search for both bounds in the range of G_j or one of these bounds. This search is basically to detect if there is a possible world containing a merge with an instance not in G_j that would give the accurate value. We detect this by following the given linkages along with their combinations.

Once again, the process for this step is based on an adaptation of Algorithm 2. The process for computing the lower bound of `min` and the upper bound of `max` is as described for the `sum`/`cnt` aggregate function. The processing for the upper bound of `min` and the lower bound of `max` starts from merges that contain single entities and moves towards merges with more entities. In all cases, the process returns the value of the aggregation function over the first accepted merge.

The complexity for computing the lower/upper value is O(|$\mathfrak{f}_i$|) when the last part of the process is not needed, otherwise the complexity is the one provided for retrieving the factor combinations since the process followed is a simple adaptation of that algorithm.

Example 6. *Consider again Q2, but with a max function (instead of sum). As in Example 5, grouping gives two sets: G_1={r_2, r_3} for "DE" and G_2={r_1,r_4,r_5} for "GR". For G_2, instance r_1 belongs to factor $\mathfrak{f}_1$, and r_4 and r_5 to $\mathfrak{f}_2$. The range for $\mathfrak{f}_1$ is [20,300] and for $\mathfrak{f}_2$ is [30,40]. For instance r_1, the value for both $\underline{v}s_{2,1}^{max}$ and $\overline{v}s_{2,1}^{max}$ is 20. We thus set the lower value to 20 and we know that the upper values is from 20 to 300. For instances r_4 and r_5, the value of $\underline{v}s_{2,2}^{max}$ is 30 and of $\overline{v}s_{2,2}^{max}$ is 40, which results in range [30,40] since its values are the same with the range values of factor $\mathfrak{f}_2$. Merging these two ranges gives a lower value 20 and upper values from 40 to 300, and after searching for the upper value (as in Example 5) the final range becomes [20,40].* ∎

5.3 Other Functions & Qualifiers

Drill-Down Qualifier. This is an optional qualifier within the `range` aggregation function. As explained earlier, an aggregation query returns tuple $\langle g, v^a, p\rangle$ for each group G_j of the given GROUP BY attribute. During query processing with a `range` aggregation function, we retrieve the range for each S_{j1}, ..., S_{jk}. We perform the same processing as `range` until the generation of these ranges. Each of these ranges is included in the answer set, and thus we have one tuple ⟨g,[$\underline{v}s_{ji}^a$-$\overline{v}s_{ji}^a$],p⟩ for each S_{ji} and for each group G_j.

Mean and Variance. Query processing with these functions is the same as DRILL-DOWN for generating the ranges per mergeable subsets of instances. It then continues by the computation of the mean or variance for group G_j based on these ranges. Given $\mu(S_{ji})$ =0.5×($\underline{v}s_{ji}^a$+$\overline{v}s_{ji}^a$), these are computed as follows:

$$mean(G_j) = \frac{\sum_{\forall S_{ji}} \mu(S_{ji}) \times mergesInRange(\underline{v}s_{ji}^a, \overline{v}s_{ji}^a)}{\sum_{\forall S_{ji}} mergesInRange(\underline{v}s_{ji}^a, \overline{v}s_{ji}^a)}, \text{ and}$$

$$variance(G_j) = \frac{\sum_{\forall S_{ji}} [\mu(S_{ji}) - mean(G_j)]^2 \times mergesInRange(\underline{v}s_{ji}^a, \overline{v}s_{ji}^a)}{\sum_{\forall S_{ji}} mergesInRange(\underline{v}s_{ji}^a, \overline{v}s_{ji}^a)}$$

Computing the total number of merges for each range is a time consuming process. We instead use an estimate that is

Algorithm 4: Evaluating an iceberg/top-k query

```
   Input: Indexing structure $I$, Query $Q$
          $P$ if this is an iceberg query or $K$ if this is a top-k query
   Output: Entities $E$
 1 $F \leftarrow$ preprocessingStep($Q$);
 2 $E \leftarrow \{\}$; // top-K entities satisfying $Q$
 3 $Temp \leftarrow \{\}$; // factors with their current probab.
 4 for ( $f_i \in F$ ) do $Temp \leftarrow Temp \cup \langle f_i, \overline{v}\mathrm{f}_i^{pr} \rangle$;
 5 i = 1;    next_prob = $\overline{v}\mathrm{f}_2^{pr}$;
 6 while (true) do
 7     $cl \leftarrow I$.getNextCombination($f_i$, $Q$);
 8     $e \leftarrow$ merge($cl$);
 9     if (e.probability<next_prob) then
10         i = i + 1;    next_prob = $\overline{v}\mathrm{f}_{i+1}^{pr}$;
11         $Temp \leftarrow Temp \setminus \{\langle f_i, \overline{v}\mathrm{f}_i^{pr} \rangle\} \cup \{\langle f_i, e.probability \rangle\}$;
12     else
13         if e.probability>P then break; // iceberg query
14         $E \leftarrow E \cup \{e\}$;
15         if E.size()<K then break; // top-k query
16 return $E$;
```

retrieved through sampling among possible worlds created by the linkages of the factor to which the specific range corresponds to. Function *mergesInRange* is replaced with $2^{|RF|} \times$ *samplesInRange*($\underline{v}s^a_{ji}$,$\overline{v}s^a_{ji}$) / *allSamples*(), where RF is the set of all distinct instances participating in the linkages of S_{ji}.

Having Qualifier. Another option is to filter the entity merges used in aggregation queries based on their probabilities. This is expressed through the HAVING optional qualifier, i.e., Figure 3, line 9. As the range probabilities are computed during query processing, the HAVING clause is applied afterwards for selecting which ranges to include/exclude from the result set. The same technique is executed also when the query contains a DRILL-DOWN.

5.4 Computing Probabilities

Query processing groups the instances satisfying the query conditions by the specified GROUP BY attribute, i.e., G_1, ... (Section 4.2). For instance, as discussed in Section 2, the result set for Q2 contains range [20-80] for location "GR" (i.e., G_1), and range [100-570] for "DE" (i.e., G_2). We denote the probability of a group as P(G_j) and compute it based on the possible worlds that can be generated for G_j given its corresponding factors. In our example, G_2 contains instances r_1, r_4, and r_5, and thus the linkages per factor are: S_{21}=$\{l_{r_1,r_2}$,$l_{r_1,r_3}\}$ and S_{22}=$\{l_{r_4,r_5}\}$. The probabilities of S_{21} and S_{22}, denoted as P(S_{ji}), are used for computing the probability of G_j. We give the details for computing P(G_j) and P(S_{ji}) in the following paragraphs.

Probability P(G_j). Assume that we have already computed the probability P(S_{ji}) for all sets S_{ji} that contain linkages with instances in group G_j. The possible worlds with entities satisfying group G_j are the ones that contain at least a merge from one of S_{j1}, ..., S_{jk}. Thus, since these linkage sets are disjoint/independent, the overall probability of the group, i.e., P(G_j), is equal to the complement of the product between the complement of all the S_{ji} probabilities that correspond to the G_j group: $P(G_j) = 1 - \prod_{S_{ji} \in G_j}(1 - P(S_{ji}))$.

Probability P(S_{ji}). We compute the probability of P(S_{ji}) as the summation of all worlds with a merge such that, when we apply the merge function, the instance returned belongs to G_j, i.e., is an answer to the query. To avoid the exponential explosion in the number of possible merges that we need to examine for computing P(S_{ji}), we do not compute the exact probability; instead, we derive an estimate based on the merges we can generate from the linkages of S_{ji} that cannot correspond to the value of the specific G_j.

More specifically, to compute P(S_{ji}), we first create a set of linkages L_{r_k} for every instance r_k from G_j. In the list, we include all the f_i linkages l_{r_α,r_β} if the merge function with the attributes of instances r_α, r_β, and r_k returns r_α or r_β. Only the linkage combination in which none of the linkages of L_{r_k} exists can be accepted. The probability for this is given by P(r_k)= 1 - $\prod_{l \in L_{r_k}}(p^l(l))$. Thus, we estimate the overall probability of as P(S_{ji}) = $\prod_{r_k \in l \in L_{r_k}} P(r_k)$, i.e., the product between the probabilities of all the possible worlds satisfying the query for G_j.

Recall that S_{ji} is the set of all linkages from factor f_i that have instances participating in group G_j. As explained above, the probability P(S_{ji}) is defined as the summation of the worlds that satisfy the given query, which is a subset of all the possible worlds that can be generated from S_{ji}. Thus, probability P(S_{ji}) is always ≤ 1.

There are approaches focusing on approximation algorithms for computing the probability bounds of query answers. For example, [25] derives bounds based on decomposing proportional formulas, [26] incrementally computes bounds given shared query plans, and [14] aims at on non-materialized views. Working with a much simpler data model (i.e., tuple-independent probabilistic databases without inner correlations) allows executing such computations. An interesting follow-up work is to investigate possible restrictions over the data model for being able to incorporate alternative approximation techniques.

6 ICEBERG AND TOP-K QUERIES

To process ICEBERG and TOP-K queries, we detect the instances satisfying the WHERE conditions (if any), and separate them into sets according to the factors, i.e., S_1, ..., S_k (Section 4.2). From the indexing structure we can also retrieve the upper/lower bound probability of each factor, i.e., [$\underline{v}\mathrm{f}_i^{pr}$, $\overline{v}\mathrm{f}_i^{pr}$] for f_i.

The indexing structure contains the top-K linkage combinations for each factor. Algorithm 4 illustrates how to evaluate an iceberg/top-k query by navigating in the structure and the factors. The algorithm uses a temporal list that contains the factors with instances satisfying the query, sorted by their upper bound probability (lines 3-4). The retrieval of the entities is iterative, and each step processes the linkage combinations in the factor with the highest probability from the temporal list. For this factor, it uses the indexing structure to retrieve combinations, and on each combination it performs its merge. These merges are included in the result list E if their probability is higher than that the upper bound of the following factor (line 12). Processing continues with the following factor (lines 9-11), when we find a merge with a probability lower than the upper bound of the following factor.

As explained in the previous sections, the indexing structure includes only the linkage combinations with the highest probabilities. This means that not all linkage combinations required by a query are included for a factor. In case this happens, we use the algorithm introduced in Section 4 to generate them.

Example 7. *As an example, let us consider query Q1 with a top-3 in the* SELECT *clause. The condition* YEAR=2010 *is satisfied by instance* r_2 *from factor* $\mathfrak{f}_1$ *and instance* r_4 *from factor* $\mathfrak{f}_2$*. We start from factor* $\mathfrak{f}_2$*, since it has the highest upper probability bound. Thus, we first retrieve instance* r_4 *with probability 0.2. We also set the current probability value to 0.2. Then, we continue with* $\mathfrak{f}_1$*, as its upper probability bound, which is 0.54, is higher than the current probability value. From* $\mathfrak{f}_1$ *we first retrieve* $e_{1,2}$ *with probability 0.36 and then* e_2 *with probability 0.1. This means that the final top-3 is given by set* $\{\langle e_{1,2}, 0.36\rangle, \langle e_4, 0.2\rangle, \langle e_2, 0.1\rangle\}$. ∎

7 Extensions

The ENTITY-JOIN is not necessarily restricted to the join between two tables, but it might contain more than two tables, either with deterministic data or with duplicates. The former means tables T_1, ..., T_k, each one referring to instances from T_{dup} and having attribute *agg_column* on which the aggregate function should be applied. Query evaluation can be performed with the introduced techniques, for instance using a new table that corresponds to the union over the *entity* and *agg_column* attributes of tables T_1, ..., T_k.

To evaluate an ENTITY-JOIN that contains two or more tables with duplicates, i.e., $T_{dup.1}$, ..., $T_{dup.2}$, we first need to create one indexing structure for each of the $T_{dup.i}$, as explained in Section 4. The basis of query evaluation would still be the techniques introduced in the paper, but query evaluation would now consider all these indexing structures instead of one indexing structure.

Another extension is to incorporate a merge function that returns an entity composed of attributes from all instances to be merged, as discussed in Section 3. This requires modification on the structure in order to operate on the attributes and not only on the instances. In short, the structure should allow the retrieval of the attributes that satisfy the query conditions, and the merge-representation edges should include the attributes that each instance "provides" in the corresponding merge.

8 Experimental Evaluation

As discussed in Section 2, there is currently no approach that enables query processing over unmerged duplicates with capabilities comparable to the ones provided by our approach. Comparing to existing approximate techniques, such as the ones discussed in Section 9, requires converting the probabilistic entity linkages to a data model supported by these approaches by partially materializing the possible worlds. Unfortunately, such conversions are not able to fully capture the semantics and the meaning encoded in the probabilistic entity linkages. Arumugam et al. [6] recently introduced a sampling-based methodology for efficiently retrieving data when probabilities are present. Their methodology can be adapted for processing queries with ENTITY-JOIN by using sampling to generate a subset of the possible worlds and evaluating the queries over them. More specifically, given a query, the method first performs pre-processing, as described in Section 4.2. Then, for every group and for every factor containing instances from that group, the method generates a number of possible worlds. Note that only possible worlds with merges satisfying the query conditions are generated. Each world is generated by randomly selecting which linkages to accept and which to reject among the linkages of the specific factor. Given factor $\mathfrak{f}_i$, we perform $c \times \ln(2^{|\mathfrak{f}_i|})$ sampling iterations where c is an iteration coefficient. This coefficient controls the quality of the sampling algorithm. Finally, the query is evaluated on the possible worlds generated by sampling, and the result sets are then used for composing the final query result set.

In the experiments, we used the **sampling-based methodology (SM)** explained above and query processing over **all possible worlds (GW)** as baselines for evaluating the **query processing over unmerged duplicates (PM)** proposed in this work. To avoid performance differences due to implementation aspects, we implemented all approaches using Java 1.6, and maintained the data in the same MySQL database. All experiments were executed on a single core of an Intel Xeon 1.6 GHz machine.

The methods were evaluated using three datasets: (1) **JaccardDB**, (2) **JaroDB**, and (3) **CoraDB**. The first two were created by integrating movie-related data coming from two popular Web systems, *IMDb* and *DBpedia*. Both datasets contained the same instances, but a different set of linkages. JaccardDB contained linkages generated by comparing the titles of IMDb and DBpedia pairwise, using *jaccard* similarity, whereas JaroDB linkages were generated using *jaro*. We used these two datasets to study the influence that the characteristics of the resolution information has on the efficiency of the proposed methodology. The datasets contained 51,221 instances (i.e., movies) with 15,529 linkages for JaccardDB and 16,835 linkages for JaroDB. For the linkage probabilities we used the results returned by the two matching methods.

The third dataset, CoraDB, contained authors and their publications from the CiteSeer system. Due to the resolution problem, the dataset does not contain a single description for each author, but various descriptions ranging between 1 and 43 and with an average 3.39 per author. This dataset contained 9,774 authors that correspond to 2,882 real-world objects. The resolution information, a total of 12,440 probabilistic linkages, was generated using a probabilistic entity linkage algorithm [20] (similar results were obtained by other proposed linkage algorithms).

We created three databases $\langle T, R, L, p^l\rangle$, one for each described dataset, as follows: the dataset instances were

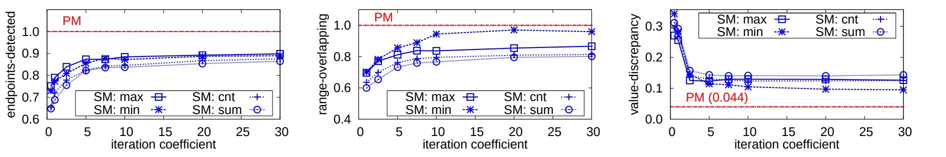


Fig. 6. Effectiveness metrics for processing queries over CoraDB vs. different sampling iterations. The two left plots are for range aggregation queries, and the right plot is for mean aggregation queries.

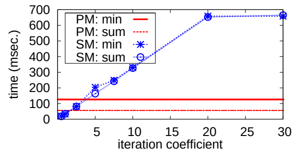


Fig. 7. Processing aggr. queries vs. sampling iterations (CoraDB).

used for R, and the linkage information for L and p^l. The last two correspond to table T_{res} included in the query syntax. The information for T was randomly generated with values following a Zipfian distribution, simulating real-world data. For JaccardDB and JaroDB, T contained movie-related information, e.g., budget and ranking. For CoraDB, T contained 4,799 positions of authors with each position including university/institute, title (e.g., "PhD student"), and position's duration. Queries were created by converting a randomly selected attribute ⟨n,v⟩ from R into WHERE conditions.

We used CoraDB in the majority of the experiments, as this is typically used for resolution algorithms. JaccardDB/JaroDB were used for studying the influence of resolution and query characteristics, as we could generate artificial queries with low selectivity (i.e., titles containing word "with").

8.1 Comparison with the GW methodology

We first compared effectiveness of PM and GW using CoraDB. As expected, for range aggregation queries both approaches returned the same results. For mean aggregation queries, where PM uses an estimate (Section 5.3), we measured the value discrepancy as $|\text{mv}_{PM} - \text{mv}_{GW}|/\text{mv}_{GW}$, where $\text{mv}_{.}$ is the mean value returned by PM or GW. The average discrepancy over 200 queries was 0.044, indicating a negligible difference.

We then compared the efficiency of PM and GW. For queries involving factors with few linkages, the time of PM had a negligible difference with the time of GW. However, the time required by GW was increasing exponentially with respect to the number of linkages. The time difference between the two approaches was 2.5 seconds when the query involved factors with 11 to 12 linkages, 44 seconds for factors with 13 to 14 linkages, 187 seconds for factors with 15 linkages, and did not finish for more than 20 linkages. These results confirm that GW is impractical for real-world data, as the ones used in our evaluation.

8.2 Comparison with the SM methodology

We now compare PM and SM. Recall that range aggregation queries return a set of tuples, each providing the range for a specific group. PM always returns the ground truth - accurate ranges (Section 8.1). However, as SM is based on sampling, the ranges it produces are either identical (in the optimal case) or subranges of the ones returned by PM. To compare the accuracy of the returned ranges, we have used two metrics. The first metric, **Endpoints-Detected**, computes the percentage of upper and lower endpoints (i.e., range values) that were correctly detected by SM with respect to the corresponding endpoints returned by PM. This occurs when we increase the sampling iterations, since the query is then evaluated over a larger number of possible worlds.

Since sampling has a lower effectiveness when dealing with skewed data, we also have used a second effectiveness metric called **Range-Overlapping**. This metric computes the overlapping between the ranges returned by SM with the ranges returned by PM. It is computed as $\text{RO}([\underline{v}^a,\overline{v}^a]_{PM}, [\underline{v}^a,\overline{v}^a]_{SM}) = \text{length}([\underline{v}^a,\overline{v}^a]_{PM} \cap [\underline{v}^a,\overline{v}^a]_{SM}) \,/\, \text{length}([\underline{v}^a,\overline{v}^a]_{PM})$, where $length([\underline{v}^a, \overline{v}^a])$ computes the length between the upper and lower value of the given range, i.e., $\overline{v}^a - \underline{v}^a$. Increasing the sampling iterations, increases the distances for the intersection of the ranges returned by PM with the ranges returned by SM, which makes the range-overlapping metric move closer to 1. For the effectiveness of mean aggregation queries, we computed the value discrepancy between the mean values returned by the two approaches, as described in Section 8.1.

Figure 6 shows the effectiveness results. Each point in the plots corresponds to the average of processing 200 queries over CoraDB. All plots report effectiveness for an increasing number of sampling iterations (i.e., $c \times \ln(2^{|\mathcal{f}_i|})$), achieved by varying the iteration coefficient c (i.e., $c \in [0.5-30]$). The two left plots correspond to range aggregation queries (endpoints-detected and range-overlapping), and the right plot to mean aggregation queries (value-discrepancy). As expected, increasing the value of c results in a better effectiveness (lower value-discrepancy, and higher endpoints-detected and range-overlapping), since then SM evaluates queries over a larger number of possible worlds. For instance, the endpoints-detected for *cnt* is 0.66 when c is 0.5, i.e., slightly more than half of the range endpoints were correctly detected by SM, and 0.87 when c is 30. In all cases, PM is substantially more accurate than SM, even when SM is configured to a high sampling coefficient.

The two approaches were also compared with respect to efficiency. The evaluation time for different sampling iterations is shown in Figure 7. With respect to SM we notice that, as expected, increasing the sampling iterations increases the evaluation time. The evaluation time for PM is constant as it does not depend on the sampling iterations. As shown in the plot, only for a very small value of c the query evaluation times of SM and PM are comparable. But, as discussed above, the accuracy of SM for such values of c is quite low.

The overall conclusion from this set of experiments is

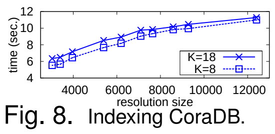


Fig. 8. Indexing CoraDB.

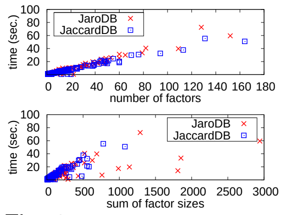


Fig. 9. Influence of resolution characteristics (movies).

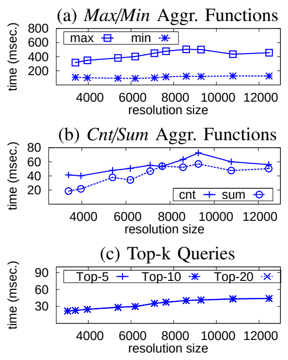


Fig. 10. Time vs. CoraDB size.

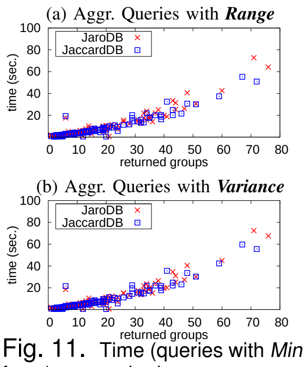


Fig. 11. Time (queries with *Min* func.) vs. resulted groups.

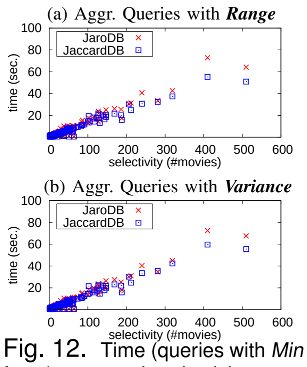


Fig. 12. Time (queries with *Min* func.) vs. query's selectivity.

that in order to reach an acceptable effectiveness with SM, one needs to move towards more sampling iterations (larger values of c), but for those values the time of SM is significantly higher than PM. For example, SM requires 0.7 seconds to evaluate a query resulting in an endpoints-detected effectiveness of 0.87 (with *min* aggregate function). Our approach requires substantially less time, and correctly returns both endpoints of the ranges.

8.3 Influence of resolution characteristics

We now explore the effect of the number of (i) linkages and (ii) factors with respect to the indexing time and query execution performance.

We first investigate the influence of the number of linkages contained in T_{res} to the indexing cost of PM. For this we used the Cora linkage set (i.e., $\mathcal{L}$) to create a small number of linkage sets (i.e., L_1, L_2, ...), each one containing a different number of linkages (i.e., $L_i \subseteq \mathcal{L}$). For each of these linkage sets, we have created a single database, and used it to evaluate the efficiency of PM. Figure 8 shows the time for creating the CoraDB indexing structure over the various resolution sizes (i.e., number of database linkages). As explained in Section 4.1, the indexing structure includes the top-K linkage combinations of each factor. The plot shows the indexing time for two different Ks, i.e., 8 & 18. The required disk space for K=8 was 3.7MB and for K=18 it was 4.2MB. As expected, increasing K also increases the required time and space, since more linkages need to be processed and included in the indexing structure. The required time scales sub-linearly with the number of linkages, allowing the indexing structure to be created in 11 seconds for the complete set of the Cora linkages. Notice that this is an one time cost, as the indexing structure will be created only once.

Figure 10 shows the query processing time for different resolution sizes, i.e., the number of T_{res} linkages. Each point in all plots is the average time for processing 400 queries. Plot (a) shows the results for ENTITY-JOIN with *min* and *max* aggregate functions, plot (b) for *cnt* and *sum* functions, and plot (c) shows the time of top-k queries for returning the Top-5, 10, & 20 answers. For all queries, we notice a small/sub-linear time increase as the resolution size increases, which occurs because query processing needs to consider a larger number of linkages.

The plots in Figure 9 show the influence of the resolution characteristics on the query processing time. For this evaluation, we processed 300 queries and measured (i) the total number of factors investigated for each query, and (ii) the total number of linkages that the factors of each query contained. As expected, the query processing time increases when the query needs to investigate more factors, or factors with more linkages. However, as discussed above, the time required by our approach is substantially better than the time required by the SM alternative methodology.

8.4 Additional aspects influencing query processing

The last set of experiments focused on investigating additional aspects that influence query processing time. We used JaroDB and JaccardDB to investigate: (i) the selectivity of the given query, and (ii) the number of groups included in the returned result set. Figures 11-12 plot the time required for each of the 300 queries with a *min* aggregate function in correlation to the number of groups included in the returned results and the selectivity of the given query. In both figures, plot (a) corresponds to aggregate queries with *range*, and plot (b) to queries with *variance*. The behavior of the approach for queries with *mean* (not in the figures) is almost identical to *variance*.

As shown by these plots, time is higher when the query has a low selectivity, or when the query returns more groups. This is expected, since in both situations query processing needs to deal with a larger number of data for constructing the results. As it typically happens, the majority of the queries has high selectivity and returns only few groups. For example, only 4 queries over JaroDB returned more than 51 groups, 6 between 41 and 50 groups, 26 between 21 and 40 groups, and the remaining queries returned up to 20 groups. With respect to selectivity, only 2 queries resulted in more than 401 movies, 21 queries between 101 and 400 movies, and the remaining up to 100 movies. This means than the majority of queries were answered in less than 20 seconds, some in less than 40 seconds, and only few required more time. In all plots, we notice that the two databases have almost the same query processing time, with JaroDB requiring slightly more time than JaccardDB. This happens because the Jaccard similarity method, used for generating JaccardDB, generated less linkages that Jaro similarity method, used for generating JaroDB, and as explained in the above evaluation, the number of linkages also affects the query processing time.

9 Related Work

The approach introduced in this work is related to two research areas, which we discuss in the following paragraphs.

A – Dealing with Entity Resolution. Our approach complements existing resolution techniques, and especially techniques that deal with unmerged duplicates.

More specifically, the approach in [31] focuses on online analytical processing. The model is based on entity clusters, from which only one can correspond to the real-world object. This model does not consider any probabilities. Query processing returns the range for the values given by an aggregate function over all possible resolutions. A similar model is followed in [18], but each cluster entry is accompanied by a belief probability. As with [31], the approach in [18] considers each entity to correspond with only one of the alternatives. Query processing returns the values of an aggregate function grouped per resulting probability, e.g., maximum 'age' for the resolved entities with probability [0.41-0.5]. In contrast to these approaches, our model is based on arbitrary probabilistic linkages between entities and focuses on efficiently supporting a more expressive and complex query syntax.

The approaches [4] and [19] investigate query processing when the detected linkages between entities are probabilistic. The former considers alternative probabilistic representations between entities, and the latter the existence of arbitrary probabilistic linkages between entities. We follow the basis of these approaches, and in particularly an extension over the [19] model (i.e., the more generic among the two models). Instead of focusing on simple singe-table queries, as [4] and [19], our query processing captures joins between the possible resolved entities and the other database tables, while also supporting qualifiers for various aggregate functions, nesting aggregation as well as qualifiers for retrieving higher level of entity details.

B – Managing Uncertain Data. This area has recently attracted the attention of the DB community [1], [22]. For example, systems such as Trio [2], MayBMS [5], PrDB [29], Orion [30], SPROUT [24], [25] and MystiQ [28], focus on data representation and efficient query processing.

In contrast to deterministic data, top-k for uncertain data has different interpretations [32]: the top-k tuples from the possible world with the highest probability, the set of k tuples that have the highest aggregated probability to appear together across all possible worlds [27], [32] (called "U-Top*k*"), and the k tuples from any possible world as long as they have the highest probabilities [32] (called "U-*k*Ranks"). The current version of our work corresponds to retrieving the *k* single-item answers with the highest probabilities (i.e., Top*k* from [27], *k* U-Top1 from [32]). Ré et al. [27] process U-Top*k* through Monte-Carlo simulation. They maintain probability intervals that are then tightened by generating random possible worlds. Soliman et al. [32] introduced a framework that navigates the space of possible worlds in order to generate the top-k tuples. More recent top-k related approaches are [26] and [14]. The approach in [26] shares the probability computation of detected subqueries with several query answer, and further extends for the computation of bounds. The goal of [14] is similar, but here the authors achieve the computation of bounds without materialization. With respect to iceberg queries, our work is based on an indexing structure that detects and maintains the entities with the highest probabilities. As such, it does not need to perform a full on-the-fly materialization, but rather directly retrieve the query answers from the indexing structure, and only, if needed, generate additional answers.

Another related category contains methods for processing joins over probabilistic data. The majority of proposed approaches are for numeric attributes [10], [17], e.g., temperature and pressure recorded by various sensors. These approaches process joins using pruning, e.g., [17] includes item-, page-, and index-pruning. There are also approaches that are not restricted to numeric attributes, such as [34], which proposes the implementation of inference algorithms (i.e., Viterbi) in-database, and based on this, achieves efficient computation of top-k probabilistic query answers.

The navigation in the indexing structure of [3] is also related to ours. In essence, in both indexes the child or of the right sibling of a node has probability equal or lower to the node. However, our work provides proofs for the provided features and additionally explains how the indexing structure can be (partially) used for evaluation of aggregation queries. Processing aggregation queries is the main goal of [16]. It is achieved by the structural decompositions of expressions into sub-expressions that are independent and mutually exclusive. Our approach supports a more expressive form of aggregation, which captures two aggregation levels. In addition, processing these aggregation levels is partially computed directly from the indexing structure.

10 Conclusions and Future Work

In this paper we address the resolution problem through a generic framework for processing complex queries over unmerged duplicates. Our approach considers a database with duplicated instances, probabilistic linkages between duplicated instances, and tables with other related data. We introduce an indexing structure that provides efficient access to the possible entity merges and their probabilities. Based on this structure, we introduce techniques for the efficient processing of aggregation and iceberg/top-k queries over the unmerged duplicates and their related data, focusing on qualifiers for retrieving analytical and summarized information. We have also performed an extensive experimental evaluation using three real-world data sets, and compared the proposed approach to a sampling-based methodology and a methodology that generates all the possible worlds. Our evaluation analyzes the resolution and query characteristics that influence the query processing time, and also verifies the approach's effectiveness and efficiency.

Our current work focuses on investigating how to extend the indexing structure for considering also the factor characteristics. As shown in the experiments (Section 8.3 and 8.4), the approach requires additional time for handling factors with a large size, i.e., factors that contain a large number of linkages. The index structure must include data according to the factor size, and this would allow balancing the time required for processing small factors with the time for processing large factors.

11 References

[1] C. Aggarwal and P. Yu. A survey of uncertain data algorithms and applications. *TKDE*, 21(5), 2009.
[2] P. Agrawal, O. Benjelloun, A. Sarma, C. Hayworth, S. Nabar, T. Sugihara, and J. Widom. Trio: A system for data, uncertainty, and lineage. In *VLDB*, 2006.
[3] M. Akdere and U. Çetintemel. Continuous probabilistic data association with constraints. Technical report, Brown University, 2011.
[4] P. Andritsos, A. Fuxman, and R. Miller. Clean answers over dirty databases: A probabilistic approach. In *ICDE*, 2006.
[5] L. Antova, C. Koch, and D. Olteanu. $10^{(10^6)}$ worlds and beyond: efficient representation and processing of incomplete information. *VLDB J.*, 2009.
[6] S. Arumugam, R. Jampani, L. Perez, F. Xu, C. Jermaine, and P. Haas. MCDB-R: Risk analysis in the database. *PVLDB*, 3(1), 2010.
[7] I. Bhattacharya and L. Getoor. Iterative record linkage for cleaning and integration. In *DMKD*, 2004.
[8] M. Bilenko, R. Mooney, W. Cohen, P. Ravikumar, and S. Fienberg. Adaptive name matching in information integration. *IEEE Intelligent Systems*, 2003.
[9] M. Bilenko, R. Mooney, W. Cohen, P. Ravikumar, and S. Fienberg. Adaptive name matching in information integration. *IEEE Intelligent Systems*, 2003.
[10] R. Cheng, S. Singh, S. Prabhakar, R. Shah, J. Vitter, and Y. Xia. Efficient join processing over uncertain data. In *CIKM*, 2006.
[11] W. Cohen, P. Ravikumar, and S. Fienberg. A comparison of string distance metrics for name-matching tasks. In *IIWeb*, 2003.
[12] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *VLDB*, 16(4), 2007.
[13] X. Dong, A. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *SIGMOD*, 2005.
[14] M. Dylla, I. Miliaraki, and M. Theobald. Top-k query processing in probabilistic databases with non-materialized views. In *ICDE*, 2013.
[15] A. Elmagarmid, P. Ipeirotis, and V. Verykios. Duplicate record detection: A survey. *TKDE*, 19(1), 2007.
[16] R. Fink, L. Han, and D. Olteanu. Aggregation in probabilistic databases via knowledge compilation. *PVLDB*, 5(5), 2012.
[17] T. Ge. Join queries on uncertain data: Semantics and efficient processing. In *ICDE*, 2011.
[18] M. Hua and J. Pei. *Ranking Queries on Uncertain Data*, chapter 7. Kluwer, 2011.
[19] E. Ioannou, W. Nejdl, C. Niederée, and Y. Velegrakis. On-the-fly entity-aware query processing in the presence of linkage. *PVLDB*, 3(1), 2010.
[20] E. Ioannou, C. Niederée, and W. Nejdl. Probabilistic entity linkage for heterogeneous information spaces. In *CAiSE*, 2008.
[21] D. Kalashnikov and S. Mehrotra. Domain-independent data cleaning via analysis of entity-relationship graph. *TODS*, 31(2), 2006.
[22] H. Kriegel, T. Bernecker, M. Renz, and A. Zuefle. *Managing and Mining Uncertain Data*, chapter 9. Springer, 2009.
[23] M. Lenzerini. Data integration: A theoretical perspective. In *PODS*, 2002.
[24] D. Olteanu, J. Huang, and C. Koch. SPROUT: lazy vs. eager query plans for tuple-independent probabilistic databases. In *ICDE*, 2009.
[25] D. Olteanu, J. Huang, and C. Koch. Approximate confidence computation in probabilistic databases. In *ICDE*, 2010.
[26] D. Olteanu and H. Wen. Ranking query answers in probabilistic databases: Complexity and efficient algorithms. In *ICDE*, 2012.
[27] C. Ré, N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *ICDE*, 2007.
[28] C. Ré and D. Suciu. Managing probabilistic data with MystiQ: The can-do, the could-do, and the can't-do. In *SUM*, 2008.
[29] P. Sen, A. Deshpande, and L. Getoor. Prdb: managing and exploiting rich correlations in probabilistic databases. *VLDB J.*, 2009.
[30] S. Singh, C. Mayfield, S. Mittal, S. Prabhakar, S. Hambrusch, and R. Shah. Orion 2.0: native support for uncertain data. In *SIGMOD*, 2008.
[31] Y. Sismanis, L. Wang, A. Fuxman, P. Haas, and B. Reinwald. Resolution-aware query answering for business intelligence. In *ICDE*, 2009.
[32] M. Soliman, I. Ilyas, and K. Chang. Top-k query processing in uncertain databases. In *ICDE*, 2007.
[33] Y. Velegrakis. On the importance of updates in information integration and data exchange systems. In *DBISP2P*, 2008.
[34] D. Wang, M. Franklin, M. Garofalakis, and J. Hellerstein. Querying probabilistic information extraction. *PVLDB*, 3(1), 2010.
[35] S. Whang, D. Menestrina, G. Koutrika, M. Theobald, and H. Garcia-Molina. Entity resolution with iterative blocking. In *SIGMOD*, 2009.

Appendix

Proof of Theorem 1: Each instance match encoded by linkage l_{r_i,r_j} exists with probability $p^l(l_{r_i,r_j})$ and does not exist with probability 1-$p^l(l_{r_i,r_j})$. Function p | $l_{r_i,r_j} \mapsto$ max($p^l(l_{r_i,r_j})$, 1-$p^l(l_{r_i,r_j})$) returns the maximum of these probabilities, and thus $p(.) \geq 0.5$.

Let function *sort* | $f_i \mapsto F_{sort}$ return a list containing the linkages of f_i sorted by the linkage probabilities, i.e., $F_{sort} = \{ l^j \mid l^j \in f_i, p(l^j) \geq p(l^{j+1}) \}$. Then, the CL_m with the highest probability among all CL_ms with m linkages not existing is as follows: $CL_m = \{l^j | l^j \in sort(f_i),\ j \leq |f_i| - m \}$.

Thus, based on Equation 2 the probability of CL_m is:

$$P(CL_m, f_i) = p((l^1) \times ... \times \mathbf{p}(l^{|f_i|-m}) \times (1 - p(l^{|f_i|-m+1})) \times ... \times (1 - p(l^{|f_i|}))$$

Similarly, the probability of CL_{m+1} is equal to:

$$P(CL_{m+1}, f_i) = p(l^1) \times ... \times p(l^{|f_i|-m-1}) \times (\mathbf{1} \text{ - } \mathbf{p}(l^{|f_i|-m})) \times ... \times (1 - p(l^{|f_i|}))$$

Therefore, $P(CL_m, f_i)/P(CL_{m+1}, f_i) = p(l^{|f_i|-m})/(1 - p(l^{|f_i|-m}))$, and since p(.)≥0.5 then $p(l^{|f_i|-m})/(1 - p(l^{|f_i|-m})) \geq 1$. For this we conclude that $P(CL_m, f_i) \geq P(CL_{m+1}, f_i)$.

Proof of Theorem 2: By definition, the probability of CL_{m+1}, i.e., $P(CL_{m+1}, f_i)$, is equal to the probability of CL_m but with one of the linkages of CL_m not included in CL_{m+1}. Therefore, the probabilities of the two combinations are:

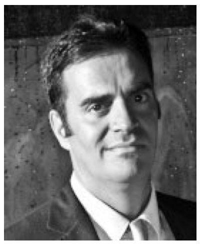
$$P(CL_{m+1}, f_i) = P(CL_m, f_i) \times [1 - p(l^j)]/p(l^j) \qquad (1)$$

$$P(CL'_{m+1}, f_i) = P(CL_m, f_j) \times [1 - p(l^{j-1})]/p(l^{j-1}) \qquad (2)$$

$$\overset{1\&2}{\Longrightarrow} \frac{P(CL_{m+1}, f_i)}{P(CL'_{m+1}, f_i)} = \frac{p(l^{j-1})}{[1 - p(l^{j-1})]} \times \frac{[1 - p(l^j)]}{p(l^j)}$$

Given that the linkages of factor f_i are sorted by probability (function *sort* in proof of Theorem 1), then $p(l^{j-1}) \geq p(l^j)$, and thus $\frac{p(l^{j-1})}{p(l^j)} \geq 1$. Also, 1-$p(l^{j-1}) \leq$ 1-$p(l^j)$, and thus $\frac{1-p(l^j)}{1-p(l^{j-1})} \geq 1$. We can thus conclude that $\frac{P(CL_{m+1},f_i)}{P(CL'_{m+1},f_i)} \geq 1$, and that $P(CL_{m+1}, f_i) \geq P(CL'_{m+1}, f_i)$.

Ekaterini Ioannou received a Ph.D. in computer science from University of Hannover, Germany. She also obtained an M.Sc. degree from Saarland University, and B.Sc. and M.Sc. degrees from University of Cyprus. Her research interests are in areas of information integration, management of uncertain data, and resolution methodologies for heterogeneous and large size collections. Currently she is a research collaborator at the SoftNet Lab, at the Technical University of Crete. During September 2012 and June 2013 she was adjunct faculty at the Open University of Cyprus.



Minos Garofalakis received a Diploma degree in Computer Engineering and Informatics (School of Engineering Valedictorian) from the University of Patras, Greece in 1992, and MSc and PhD degrees in Computer Science from the University of Wisconsin-Madison in 1994 and 1998, respectively. He worked as a Member of Technical Staff at Bell Labs, Lucent Technologies in Murray Hill, NJ (1998-2005), as a Senior Researcher at Intel Research Berkeley in Berkeley, CA (2005-2007), and as a Principal Research Scientist at Yahoo! Research in Santa Clara, CA (2007-2008). In parallel, he also held an Adjunct Associate Professor position at the EECS Department of the University of California, Berkeley (2006-2008). As of October 2008, he is a Professor of Computer Science at the School of Electronic and Computer Engineering of the Technical University of Crete, and the Director of the Software Technology and Network Applications Laboratory (SoftNet).