Analytics over Probabilistic Unmerged Duplicates

Ekaterini Ioannou and Minos Garofalakis

Technical University of Crete, Chania, Greece
ioannou,minos@softnet.tuc.gr

Abstract. This paper introduces probabilistic databases with unmerged duplicates (DB^{ud}), i.e., databases containing probabilistic information about instances found to describe the same real-world objects. We discuss the need for efficiently querying such databases and for supporting practical query scenarios that require analytical or summarized information. We also sketch possible methodologies and techniques that would allow performing efficient processing of queries over such probabilistic databases, and especially without the need to materialize the (potentially, huge) collection of all possible deduplication worlds.

Keywords: probabilistic databases, unmerged duplicates, query analytics.

1 Introduction

Entity Deduplication 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**. Traditional deduplication techniques [4] 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. Query processing is performed over the updated data set.

The newly introduced deduplication techniques, for example [1], [6] and [9], aim at addressing the challenges appearing in modern applications (e.g., Web 2.0). These challenges primarily involve higher levels of heterogeneity and more frequent data modifications [11]. Instead of creating the entities using a-priori merging of instances, these approaches maintain and use *unmerged duplicates*, indicating which instances describe the same real-world entities. The unmerged duplicates are maintained alongside the original data and are used during query processing for generating answers that reflect the different possible real-world scenarios, i.e., deduplication decisions.

Existing techniques incorporate unmerged duplicates of different forms, such as clusters of instances that describe the same real-world object [1, 9], and linkages between pairs of matching instances [6]. In some techniques, the unmerged duplicates are accompanied by probabilities that encode the inherent uncertainty of the deduplication process. These techniques [1, 6, 9] focus on processing *simple (single-table) queries* over the unmerged duplicates.

Although answering simple queries over unmerged duplicates is important, it is still just a first step towards a complete solution to in-database entity deduplication. The

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

Deduplication

id	*inst_1*	*inst_2*	*pr*
l_{r_1,r_2}	r_1	r_2	0.95
l_{r_1,r_3}	r_1	r_3	0.55

Order

id	*buyer*	*items*	*amount*
t_1	r_1	1	100
t_2	r_2	2	300
t_3	r_2	4	250
t_4	r_3	2	250

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

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 deduplication scenarios can easily overwhelm the user, as the number of these scenarios is huge [2]. 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.

In this paper, we introduce DB^{ud}: a database containing probabilistic information about instances found to describe the same real-world objects. DB^{ud} adopts the most expressive form of deduplication information (i.e., probabilistic linkages between instances – also accounting for transitivity), and significantly extends its scope by considering the deduplication information as part of a database with other tables providing entity-related data. In the following sections, we first introduce analytical queries for retrieving information of the entities in DB^{ud} (Section 2), and then sketch possible methodologies and techniques for efficiently processing queries over such a probabilistic database with unmerged duplicates (Section 3).

2 Modeling Data and Queries

A probabilistic database with unmerged duplicates DB^{ud} contains deterministic relational tables T_1, ..., T_n as well as tables with duplicates R_1, ..., R_k, i.e., some instances of R_i describe the same real-world objects. The deduplication information for table R_i is given in table L_i. More specifically, L_i contains probabilistic linkages over the instances in R_i: $l_{r_\alpha,r_\beta} \in L_i$ means that instances r_α and r_β from R_i describe the same real-world object with probability p^l.

To process queries over DB^{ud} we must be able to support joins between the tables with unmerged duplicates and the deterministic tables. For example, answering queries over the DB^{ud} fragment shown in Figure 1 requires considering the join between table Buyer with Order. Since table Buyer contains duplicates, we must first derive the possible entities using the deduplication information provided in table Deduplication. Each linkage from the Deduplication table can be either accepted or rejected, e.g., we can accept l_{r_1,r_3} with probability 0.55 or reject it with probability (1-0.55). Rejecting the linkage means that the database has two entities, one for each of the instances. Accepting the linkage implies a new entity, with identifier $e_{1,3}$, that replaces both r_1 and r_3. Creating a single entity given these two instances maybe performed using different semantics. For example, if we assume that we keep the instance with the highest value on

	Linkages	Prob.	Entities (with summation over Order's amount)
I_1	l_{r_1,r_2} & l_{r_1,r_3}	0.5225	$\langle e_{1,2,3}$, ..., 2009, DE, 900$\rangle$
I_2	l_{r_1,r_2} & $\neg l_{r_1,r_3}$	0.4275	$\langle e_{1,2}$, ..., 2010, DE, 650$\rangle$, $\langle e_3$, ..., 2011, DE, 250$\rangle$
I_3	$\neg l_{r_1,r_2}$ & l_{r_1,r_3}	0.0275	$\langle e_{1,3}$, ..., 2011, DE, 800$\rangle$, $\langle e_2$, ..., 2010, DE, 550$\rangle$
I_4	$\neg l_{r_1,r_2}$ & $\neg l_{r_1,r_3}$	0.0225	$\langle e_1$, ..., 2009, GR, 100$\rangle$, $\langle e_2$, ..., 2010, DE, 550$\rangle$, $\langle e_3$, ..., 2011, DE, 250$\rangle$

Table 1. The possible deduplication worlds with the entities created when requesting the join between Order with Buyer and summation over the Order's "amount" for each entity.

the year attribute, the tuple for the merge between instances r_1 and r_3 is $\langle e_{1,3}$, "Mary", "Smith", "DE", "female", "2011"$\rangle$.

For creating the possible entities of a table with unmerged duplicates R_i we need to consider the acceptance and rejection of each linkage of L_i. Deciding which linkages from L_i (e.g., table Deduplication from Figure 1) to accept or reject leads to a huge (exponentially-large) number of situations, termed *possible deduplication worlds*. Generating all these situations is infeasible. In addition, the huge volume of results that would arise when processing queries over all possible worlds 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 deduplication world*, using conventional SQL aggregate semantics over the merged entities. For example, consider again the data from Figure 1. Accepting both linkages of table Buyer leads to entity $e_{1,2,3}$, which would join with tuples t_1, t_2, t_3 and t_4 from table Order. The summation over the Order's "amount" is thus 900. Table 1 shows the four deduplication worlds created when requesting the join between Order and Buyer with summation over the Order's "amount" for each entity.

Note that we also need to identify and ignore the deduplication worlds in which the entities created by the accepted linkages are not satisfied by the rejected linkages. For example, the deduplication world with entity $e_{\alpha,\beta,\gamma}$ is invalid if it was created by accepting the linkages $l_{\alpha,\beta}$ and $l_{\alpha,\gamma}$ and rejecting linkage $l_{\beta,\gamma}$.

Although the first aggregation level leads to 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 propose considering a **second aggregation level**, *across all possible deduplication worlds*, over all the records created by the first level and based on one (or more) query attributes of interest. These probabilistic, second-level aggregation semantics, can express and evaluate queries with different analytical operators over the collection of possible worlds. We now describe two examples.

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:

SELECT Buyer.location, range(entity_amount), prob
FROM Order **entity-join** Buyer **based on** Deduplication
using sum(Order.amount) as entity_amount
WHERE GROUP BY Buyer.location

Although not directly expressed in the query, the ENTITY-JOIN 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). Evaluating the (explicit) GROUP BY clause over the resulting records gives two locations: "GR" and "DE" (aggr. level 2). Consider now all entities in the possible worlds, i.e., I_{1-4} of Table 1. The amount summation for location "GR" is 100, and for location "DE" it is between 250 and 900, and thus the range is [250-900]. The probability for each location is the summation of the possible worlds in which they participate. The location-range pairs along with their probabilities that compose the answer set are: {⟨"GR", [100-100], 0.0225⟩, ⟨"DE", [250-900], 1⟩}.

Another useful query type is iceberg that allows users to find the *high-probability deduplication scenarios* satisfying specific selection predicates. As an example, consider the following "iceberg" query:

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

This query aims at computing the two most likely aggregate amounts spent by buyers in 2010, along with their respective probabilities. The entities satisfying the WHERE conditions are e_2 from possible worlds I_3 and I_4, $e_{1,2}$ from I_2. The probability of each entity is summation of the probabilities of the worlds in which it participate, i.e., 0.05 for e_2 and 0.4275 for $e_{1,2}$. By default, the entities are ordered by probability, thus, the answer for this query is {⟨650, 0.4275⟩, ⟨550, 0.05⟩}.

Our vision is to provide complex aggregation and iceberg queries that will allow users to efficiently retrieve statistical information about the possible deduplicated entities. As shown in the above examples, a vital operator is a novel ENTITY-JOIN, which will allow expressing joins between a table with unmerged duplicates R_i and deterministic database table T_j. Entities are created using `summation`, `count`, `minimum`, or `maximum` aggregation over the T_j tuples. The ENTITY-JOIN can be used for query analytics using either aggregation operators (e.g., `range`, `mean` and `variance`[1]) or iceberg operators (e.g., top-k). Instead of top-k, we could also consider simply specifying a lower bound on the probability of the returned aggregate values.

Users might also be interested in retrieving results with more details, probably after executing aggregation queries, which basically implies reversing parts of the performed summarization. This can be performed with a *"drill down"* qualifier, similar to the corresponding qualifier of online analytical processing.

Providing efficient operators for constructing entities given a set of instances is also useful for query processing over DB^{ud}. The majority of the existing deduplication approaches either do not deal with this issue or simply return the most recent instance or the union of all instances. To provide such operators, we could for example consider the [12] approach from information extraction, which constructs entities by detecting a canonical value for each attribute given the corresponding values from all the instances.

[1] `Mean` can be used for retrieving the average value over the ranges of all possible merges and `variance` for indicating the typical discrepancy of the expected value.

3 Possible Mechanisms for Efficient Query Processing

For providing analytics over DB^{ud}, we need to introduce new mechanisms and techniques that exploit processing of aggregation and iceberg queries without the need to materialize the possible worlds. Other important aspects that we must consider, include the efficient computation of probabilities over the resulting answers, and the linkage transitivity requirement that, among other things, implies the need for reasoning at query time.

Aggregation queries. This type of queries has been so far studied only by very few approaches. For example, processing aggregation queries is the main goal of [5]. It is achieved by the structural decompositions of expressions into sub-expressions that are independent and mutually exclusive. DB^{ud} needs to support a more expressive form of aggregation, which captures two aggregation levels.

Another existing approach that targets aggregate operators is [9]. However, there exist crucial differences with the aggregate operators required for DB^{ud}. One difference is that the model followed in [9] assumes that the algorithm is provided with fixed clusters of instances, which allows focusing on basic query-time aggregation. In sharp contrast to [9], DB^{ud} follows a more generic deduplication model that requires dealing also with linkages between instances as well as linkage transitivity. In addition, DB^{ud} also considers probabilistic linkages, in order to capture the relevant entity-linkage uncertainty. Another difference is that DB^{ud} supports a more expressive query syntax in comparison to [9], which includes two aggregation levels and additional aggregation functions.

Processing aggregation queries over DB^{ud} could be efficiently achieved by limiting the number of possible worlds to be materialized or by partially materializing possible worlds. For instance, for minimum and maximum aggregates we do not need to use all the records but rather only one record from T_i for each instance from R_i. As an example, consider again the data of Order from Figure 1. When processing a query with a maximum aggregate, we can safely ignore all tuples related to a specific r_i except the one with the highest amount, i.e., for r_2 we keep only tuple t_2 since this provides the highest amount among all tuples related to r_2.

Iceberg queries. In contrast to deterministic data, iceberg queries (i.e., top-k) for uncertain data has different interpretations [10]: 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 [8, 10] (called "U-Top$k$"), and the k tuples from any possible world as long as they have the highest probabilities [10] (called "U-kRanks"). For DB^{ud}, this query type corresponds to retrieving the k single-item answers with the highest probabilities (i.e., Top$k$ from [8], k U-Top1 from [10]). Ré et al. [8] process U-Top$k$ through Monte-Carlo simulation. They maintain probability intervals that are then tightened by generating random possible worlds. Soliman et al. [10] 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 [7] and [3]. The approach in [7] shares the probability computation of detected subqueries with several query answer, and further extends for the computation of bounds. The goal of [3] is similar, but here the authors achieve the computation of bounds without materialization.

One option for processing iceberg queries over DB^{ud}, is to create an indexing structure that detects and maintains the entities with the highest probabilities. Ideally, the

indexing structure would provide efficient access to the information encoded through the linkages (i.e., potential merges) and allow easy construction of possible worlds (or partial possible worlds), as well as the fast retrieval of their probabilities. Thus, DB^{ud} would not need to perform a full on-the-fly materialization, but rather directly retrieve query answers, or part of them, from the indexing structure.

4 Summary

In this paper we have presented probabilistic databases with unmerged duplicates, i.e., databases with duplicated instances and probabilistic linkages between duplicated instances. We discussed the need for efficiently supporting practical query scenarios that do not require retrieving the huge collection of all possible deduplication worlds, but rather analytical or summarized information. This primarily involves query analytic, including aggregation and iceberg queries. We have also sketch possible methodologies and techniques that would allow the efficient processing of queries over such probabilistic databases, and especially without the need to materialize the collection of all possible deduplication worlds.

References

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

[2] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *VLDB*, 16(4), 2007.

[3] M. Dylla, I. Miliaraki, and M. Theobald. Top-k query processing in probabilistic databases with non-materialized views. In *ICDE*, 2013.

[4] A. Elmagarmid, P. Ipeirotis, and V. Verykios. Duplicate record detection: A survey. *TKDE*, 19(1), 2007.

[5] R. Fink, L. Han, and D. Olteanu. Aggregation in probabilistic databases via knowledge compilation. *PVLDB*, 5(5), 2012.

[6] 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.

[7] D. Olteanu and H. Wen. Ranking query answers in probabilistic databases: Complexity and efficient algorithms. In *ICDE*, 2012.

[8] C. Ré, N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *ICDE*, 2007.

[9] Y. Sismanis, L. Wang, A. Fuxman, P. Haas, and B. Reinwald. Resolution-aware query answering for business intelligence. In *ICDE*, 2009.

[10] M. Soliman, I. Ilyas, and K. Chang. Top-k query processing in uncertain databases. In *ICDE*, 2007.

[11] Y. Velegrakis. On the importance of updates in information integration and data exchange systems. In *DBISP2P*, 2008.

[12] M. Wick, K. Rohanimanesh, K. Schultz, and A. McCallum. A unified approach for schema matching, coreference and canonicalization. In *KDD*, 2008.