On entity resolution in probabilistic data

Ayat, S.N.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: https://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.

UvA-DARE is a service provided by the library of the University of Amsterdam (http://dare.uva.nl)
Chapter 5
Entity Resolution for Probabilistic Data Using Entropy Reduction

5.1 Introduction

One of the aspects associated with the entity resolution (ER) problem (as discussed in Chapter 1) is the deduplication. In this chapter, we deal with the deduplication definition of the ER problem (see Definition 1.2.2).

The ER problem arises in many applications that need to deal with probabilistic data. Let us provide an example of such applications from the anti-crime domain.

5.1.1. Example. ER on police’s criminal records. The anti-crime police is faced with many new crimes every year. It spends lots of time and money gathering data about every crime from different sources such as witnesses, interrogations, police’s informants, and reconstruction of the crime scene. Most of the gathered data are however not certain. For instance, the police cannot completely trust informants and witnesses, or is not sure about the information gathered by reconstructing the crime scene. Thus, the confidence values can be attached to the gathered and stored data to show their likelihood of truth, according to the confidence on the sources. These probabilistic data can in turn help to speed up the investigation process, in solving the open cases, and in finding suspects in later crimes. In a simplified form, the police maintains a single relation Criminals that contains data about all criminals. In this relation, each individual criminal is represented by an entity that consists of a number of alternative tuples, each associated with a probability value showing its likelihood of truth. It happens quite often that a criminal participates in several crimes. Such a criminal is then represented with multiple entities in the Criminals relation. Being able to find and merge such entities, which in fact refer to the same criminal, helps the police in obtaining

\[1\] The material of this chapter has been partially published in [20].
more information about the criminals and can greatly speedup the investigation process.

In information theory, the amount of uncertainty in a random variable represents its quality. This means that the more uncertain the random variable, the less predictable its outcome, and thus the lower its quality. Also, in a probabilistic database, the amount of uncertainty in the database represents its quality, meaning that the more uncertain the database, the lower its quality and thus the quality of the query results over it. For better intuition, consider a tuple $t$ with existence probability $p(t)$. If $p(t)$ is close to zero (or one), we can say with high probability that $t$ does not exist (or exists) in the database, meaning that we have minimum uncertainty about $t$. On the other hand, if $p(t)$ is close to 0.5, the probability that $t$ exists and the probability that $t$ does not exist in the database are almost the same, meaning that we have maximum uncertainty about $t$. Entropy is a well known metric for measuring the amount of uncertainty in a random variable [119], since the entropy of a random variable increases with its uncertainty. For instance, the entropy of tuple $t$ where $p(t)$ is close to zero or one, is close to zero (i.e. minimum uncertainty) and the entropy of $t$ where $p(t)$ is close to 0.5, is close to one (i.e. maximum uncertainty).

In this chapter, we use entropy as a quality metric for measuring the quality of a probabilistic database. The aim of ER is to improve the quality of the database and thus to improve the quality of the query results over the database. Thus, in the ER process, we take the entropy reduction as a powerful tool for deciding about the probabilistic tuples that should be merged.

To deal with the problem of ER over probabilistic data (denoted by ERPD) using entropy, we need a solution that efficiently: 1) computes the entropy in probabilistic databases; 2) merges probabilistic tuples that should be merged; and 3) produces a cleaned database with (near) minimum entropy. In this chapter, we propose such a solution. To the best of our knowledge, this is the first proposed solution for efficient ER over probabilistic databases using entropy reduction. Our contributions are summarized as follows:

- We model the ERPD problem as an entropy minimization problem.
- We propose an efficient technique for computing the entropy of a probabilistic database in the x-relation model [10].
- We propose a merge function, denoted by CAF, for merging probabilistic tuples and entities (i.e. x-tuples in the x-relation model).
- We propose an efficient algorithm, denoted by ME, that uses our proposed merge function to deal with the ERPD problem by producing a cleaned database with (near) minimum entropy.
5.2 Problem Definition

We also evaluate the performance of our approach through experimentation over both real-world and synthetic data. The results show that our approach is scalable in both the number of tuples in the database and the average number of duplicate tuples per entity. The experimental results also show that our proposed approach can significantly reduce the uncertainty of the database and improve the quality of the results of queries over it. Our algorithm significantly outperforms the best existing algorithms for ERPD by factors up to 144.

The rest of this chapter is structured as follows. In Section 5.2, we provide some background on the probabilistic data model that we use and entropy in probabilistic databases, and then we precisely define the problem addressed in this chapter. Section 5.3 presents an efficient method for computing the entropy of a probabilistic database in the x-relation model, the CAF merge function, and the ME algorithm. In Section 5.4, we evaluate the performance of our approach over both synthetic and real-world databases. In Section 5.5, we analyze our approach against related work, and Section 5.6 concludes the chapter.

5.2 Problem Definition

In this section, we first describe the probabilistic data model that we adopt. Then, we define entropy in probabilistic databases. Finally, we formally state the problem which we address.

5.2.1 Data Model

For representing an uncertain database, we use the x-relation probabilistic data model in which each uncertain entity is represented with an x-tuple (see the definition in Section 2.1.2). Figure 5.1(a) shows an example database $D$ in the x-relation model. This database consists of two x-tuples $x_1$ and $x_2$, where $x_1$ consists of two alternatives $t_1$ and $t_2$, and $x_2$ consists of only one alternative $t_3$.

We denote an uncertain database by $D$, the set of its possible worlds by $PW(D)$, and the set of all tuples in $D$ by $D$. We assume that the sum of the probabilities of all alternatives of an x-tuple, say $sum_p$, is equal to 1. If $sum_p$ is less than 1, we conceptually add a null tuple, denoted by $t_{\perp}$, with probability $1 - sum_p$ to the x-tuple. This tuple is only used for completeness in proofs and does not exist physically. For instance, in Figure 5.1(a), the probability of the null tuple of $x_1$ is 0.1 and that of $x_2$ is 0.8.

5.2.2 Entropy in Probabilistic Databases

In information theory, entropy is a measure of the uncertainty associated with a random variable [119]. It is usually used for quantifying the expected uncertainty (or quality) of communicated information. In this chapter, we use entropy to
5.2.1. Definition. Entropy of a probabilistic database. Let \( D \) be a probabilistic database, \( PW(D) \) be the possible worlds of \( D \), and \( p(w) \) be the probability of a possible world \( w \). The entropy of \( D \) is defined as

\[
\text{entropy}(D) = - \sum_{w \in PW(D)} p(w) \cdot \log p(w), \tag{5.1}
\]

Notice that the base of the log function is 2. As an example, consider the probabilistic database \( D \) and its possible worlds in Figure 5.1. Then, using equation (5.1), \( \text{entropy}(D) \) is 2.02.

5.2.3 Entity Resolution Over Probabilistic Database

Like in deterministic databases, the ER problem in probabilistic databases contains two main phases: 1) duplicate detection; 2) merging duplicates. Since the probabilities of tuples are irrelevant in detecting the duplicate tuples, we can ignore them and use one of the existing proposals for duplicate detection over deterministic data.

However, the merging phase of ERPD is different from that of ER, because the merge function in ERPD should adjust the probabilities of the tuples to satisfy the constraints of the probabilistic data model. For instance, consider two non-resolvable conflicting duplicate tuples \( t_1 \) and \( t_2 \) with probabilities 0.8 and 0.3 respectively, in the x-relation model. These tuples cannot be merged into a single
5.2. Problem Definition

tuple, thus, the merge function decides to put them in an x-tuple and return it as the result of the merge. However, the sum of the probabilities of \( t_1 \) and \( t_2 \) is more than 1, thus, the merge function should adjust \( p(t_1) \) and \( p(t_2) \) to satisfy the constraint \( p(t_1) + p(t_2) \leq 1 \) before putting them in the output x-tuple.

The way that the merge function merges the duplicate tuples and adjusts the tuples probabilities greatly depends on the application domain and the interpretation of the probabilities. To enable using custom merge functions, we consider the merge function as a black-box which accepts two x-tuples as input and merges them into an output x-tuple.

In this chapter, we distinguish between the merge function and the merge phase of ERPD, where the merge phase is the process that chooses the x-tuples to be merged and calls the merge function to merge them.

In the x-relation model, since the duplicate tuples are in fact different representations or beliefs about the same real-world entity, it is reasonable to expect a mutual exclusion relation between them in the cleaned database. Thus, ideally, the merge phase returns an x-tuple as the result of merging the duplicate tuples of an entity.

However, there are situations where merging all duplicate tuples of an entity into an x-tuple increases the entropy, which is in contrast to the aim of ERPD. For instance, consider an entity which has two duplicate tuples \( t_1 \) and \( t_2 \) with non-resolvable conflicts, where \( p(t_1) = p(t_2) = 0.9 \), and a merge function which adjusts \( p(t_1) \) and \( p(t_2) \) to 0.47 and returns the x-tuple \( \{t_1, t_2\} \) as the result of the merge. Now, while the entropy of the original tuples is about 0.94, the entropy of the resulted x-tuple is about 1.24, which shows an increase of 0.3 in the entropy.

To avoid such situations, we have to relax the assumption of merging all duplicate tuples of an entity into an x-tuple. Thus, the merge phase may merge the duplicate tuples in different combinations. For example, suppose \( M \) is a merge function, and \( t_1, t_2, \) and \( t_3 \) are three duplicate tuples. Let \( \{t_i\} \) denote the x-tuple with the single alternative \( t_i \), and \( t_i \odot t_j \) denote the tuple resulted from merging \( t_i \) and \( t_j \). The merge phase may then produce the following results:

1. \( \{t_1\}, \{t_2\}, \{t_3\} \)
2. \( \{t_1 \odot t_2\}, \{t_3\} \)
3. \( \{t_1 \odot t_3\}, \{t_2\} \)
4. \( \{t_2 \odot t_3\}, \{t_1\} \)
5. \( \{(t_1 \odot t_2) \odot t_3\} \)

5.2.4 Problem Statement

In this chapter, we aim at merging the x-tuples while minimizing the entropy. Since we define the entropy as a measure of uncertainty in probabilistic databases,
we expect that the result of the merge phase be a database with minimum uncertainty, thus with high quality.

We then define the ERPD problem as an entropy minimization problem where given a probabilistic database, we should produce a cleaned database with minimum entropy. Below, we formally define the ERPD problem.

5.2.2. Definition. ERPD problem. Let $D$ be an $x$-relation from the tuple domain $D$. Let $E = \{E_1, \ldots, E_m\}$ be a partitioning of $D$ tuples, where each partition is believed to contain duplicate tuples representing the same real-world entity. Let $M$ be a merge function that merges two $x$-tuples into an output $x$-tuple. Let $C_i = \{C_{i,1}, \ldots, C_{i,k}\}$ be a partitioning of $E_i$ tuples, where $E_i \in E$. Let $x_{i,j}$ be the $x$-tuple resulted from merging all tuples in $C_{i,j} \in C_i$, using the merge function $M$, and $X_i = \bigcup_{j=1}^{k} x_{i,j}$. Let $D^c = \bigcup_{i=1}^{m} X_i$. Then, we define the entropy minimized cleaned database $D^c_{\text{min}}$ as

$$D^c_{\text{min}} = \arg \min_{D^c} \text{entropy}(D^c).$$

The ERPD problem is to find $D^c_{\text{min}}$.

A naïve solution for the above problem is to enumerate all possible partitionings for each set $E_i$ of duplicate tuples and find the one with minimum entropy. However, this solution is inefficient because the number of different partitionings to be examined is exponential to the number of duplicate tuples in each set $E_i$. Given an $x$-relation $D$, our aim in this chapter is to efficiently find $D^c_{\text{min}}$.

5.3 ERPD Using Entropy

In this section, we propose our solution for the ERPD problem based on entropy minimization. We first describe our approach for computing the entropy of a probabilistic $x$-relation database. Then, we deal with merging probabilistic entities, using a merge function, called CAF. Then, by using CAF, we propose an algorithm that can efficiently approximate the entropy minimized cleaned database with a controlled error bound. Finally, we analyze this algorithm’s complexity.

5.3.1 Computing Entropy in X-relations

In our ER technique, we need to compute the entropy of probabilistic databases. By Definition 5.2.1 (in Section 5.2.2), we defined the entropy of a probabilistic database based on the probability of its possible worlds. However, the difficulty with a direct utilization of this definition is that we have to enumerate all possible worlds, which can be exponentially large.

Fortunately, if the probabilistic database is in the $x$-relation model, we can efficiently compute the entropy. By the following lemma, we propose the basis for efficient computation of entropy in $x$-relations.
5.3. ERPD Using Entropy

5.3.1. Lemma. Let $\mathcal{D} = \{x_1, \ldots, x_k\}$ be a probabilistic database in the $x$-relation model where $x_i, i \in [1..k]$ denotes the $i$th $x$-tuple of $\mathcal{D}$. Then, $\text{entropy}(\mathcal{D})$ is equal to

$$
\text{entropy}(\mathcal{D}) = - \sum_{x \in \mathcal{D}} \sum_{t \in x} p(t). \log p(t)
$$

(5.2)

Proof. The proof is by induction on the number of $x$-tuples in the database. Basis ($\mathcal{D} = \{x\}$): considering the fact that $\text{PW}(\mathcal{D}) = \{\{t\} \mid t \in x\}$ and using (5.1), we have

$$
\text{entropy}(\mathcal{D}) = - \sum_{w \in \text{PW}(\mathcal{D})} p(w). \log p(w) = - \sum_{t \in x} p(t). \log p(t) = \text{RHS}(5.2)
$$

Inductive step: let us assume that $\mathcal{D} = \{x_1, \ldots, x_k\}$ and $\text{entropy}(\mathcal{D})$ is equal to $- \sum_{t \in \mathcal{D}} \sum_{t \in x} p(t). \log p(t)$, for some $k > 1$. We show that equation (5.2) holds for database $\mathcal{D}' = \mathcal{D} \cup x_{k+1}$, where $x_{k+1}$ is an arbitrary $x$-tuple. Using (5.1) we have:

$$
\text{entropy}(\mathcal{D}') = - \sum_{w' \in \text{PW}(\mathcal{D}')} p(w'). \log p(w')
$$

(5.3)

Since $x$-tuples occur independently in the $x$-relation model, we have:

$$
\text{PW}(\mathcal{D}') = \{w' \mid w' = w \cup \{t\}, w \in \text{PW}(\mathcal{D}), t \in x_{k+1}\}
$$

(5.4)

Rewriting equation (5.3) using equation (5.4) and considering the fact that $p(w') = p(w).p(t)$, we have

$$
\text{entropy}(\mathcal{D}') = - \sum_{t \in x_{k+1}} \sum_{w \in \text{PW}(\mathcal{D})} (p(t).p(w). \log (p(t).p(w))) =
$$

$$
- \sum_{t \in x_{k+1}} \sum_{w \in \text{PW}(\mathcal{D})} p(t).p(w). \log p(t) + \sum_{t \in x_{k+1}} \sum_{w \in \text{PW}(\mathcal{D})} p(t).p(w). \log p(w) =
$$

$$
- \sum_{t \in x_{k+1}} p(t). \log p(t) \cdot \sum_{w \in \text{PW}(\mathcal{D})} p(w) + \sum_{w \in \text{PW}(\mathcal{D})} \sum_{t \in x_{k+1}} p(t).p(w). \log p(w) =
$$

$$
- \sum_{t \in x_{k+1}} p(t). \log p(t) \cdot \sum_{w \in \text{PW}(\mathcal{D})} p(w) + \sum_{w \in \text{PW}(\mathcal{D})} p(w). \log p(w) \cdot \sum_{t \in x_{k+1}} p(t)
$$

(5.5)

Substituting $1$ as the value of $\sum_{w \in \text{PW}(\mathcal{D})} p(w)$ and $\sum_{t \in x_{k+1}} p(t)$ in (5.5), we have

$$
\text{entropy}(\mathcal{D}') = - \sum_{t \in x_{k+1}} p(t). \log p(t) + \sum_{w \in \text{PW}(\mathcal{D})} p(w). \log p(w) =
$$

$$
- \sum_{t \in x_{k+1}} p(t). \log p(t) + \text{entropy}(\mathcal{D})
$$
Using induction assumption, we have

\[
\text{entropy}(\mathcal{D}') = - \sum_{t \in \mathcal{D}_{k+1}} p(t) \log p(t) + \sum_{x \in \mathcal{D}} \sum_{t \in x} p(t) \log p(t) = \\
- \sum_{x \in \mathcal{D}'} \sum_{t \in x} p(t) \log p(t) = RHS(5.2).
\]

Using Lemma 5.3.1, we can efficiently compute the entropy of a database in the x-relation model in \(\theta(n)\) time, where \(n\) is the number of tuples in the database.

### 5.3.2 Merge Function

In addition to a method for computing entropy, our ER technique needs a merge function for merging x-tuples. For this, we first consider the problem of merging two tuples. Then, we explain the way that a generic merge function merges two x-tuples, and propose the CAF merge function which has the properties that enable us to efficiently deal with the ERPD problem.

#### Merge of Tuples

We consider the value of an attribute in a tuple as the belief of the tuple about that attribute. Having null, denoted by symbol \(\perp\), as the value of an attribute means the tuple has no belief in that attribute. We also consider a tuple as the intersection of beliefs in its individual attributes. More precisely, consider tuple \(t = (v_1, \ldots, v_n)\) on schema \(S = (A_1, \ldots, A_n)\). Let \(b_i\) be the belief that \(A_i = v_i\). Then, tuple \(t\) can be considered as \(\cap_{i=1}^{n} b_i\).

We define the merge of two tuples, denoted by \(\odot\), as the intersection of their beliefs about their individual attributes. From the point of view of merging, we consider two types of tuples: mergeable and non-mergeable tuples. Two tuples, say \(t\) and \(t'\), are mergeable if they have the same beliefs about all of their individual attributes. In other words, they do not have attributes with conflicting values. If two tuples cannot be merged, then we say that they are non-mergeable. Notice that null values do not conflict with each other and also with other attribute values. In mergeable tuples, the existence of null values in each of the two tuples causes the merge result to be not equal to one or both of the tuples. We consider three main categories for mergeable tuples:

- **Mergeable type 1**: the merge result is not equal to any of the two tuples.
- **Mergeable type 2**: the merge result is only equal to one of the two tuples.
- **Mergeable type 3**: the merge result is equal to both tuples.
Notice that $t_\perp$ is mergeable with every tuple and the result of the merge is equal to that tuple. In other words, $t_\perp$ is the neutral element of $\odot$ operation.

Let us illustrate the mergeable, non-mergeable, and the three mergeable types using an example.

**5.3.2. Example.** Consider tuples $t_1, \ldots, t_5$ in Figure 5.2(a). $t_1$ is non-mergeable with the other tuples except $t_3$, since they conflict on the value of the address attribute. $t_2, \ldots, t_5$ are mergeable, since they do not conflict on the value of any attribute. Notice that different representations, e.g. “Thomas Michaelis” and “T. Michaelis”, are not considered as conflicting values. $t_2$ and $t_3$ are mergeable type 1, since $t_2 \odot t_3 \neq t_2 \neq t_3$. $t_2$ and $t_4$ are mergeable type 2, since $t_2 \odot t_4$ is equal to $t_4$ but not equal to $t_2$. $t_4$ and $t_5$ are mergeable type 3, since $t_4 \odot t_5 = t_4 = t_5$.

**CAF Merge Function**

Our ER algorithm, which we present in Section 5.3.3, requires that the merge function is commutative and associative, to be able to merge x-tuples in any order. To meet this requirement, we have developed a merge function, denoted as CAF (i.e. Commutative Associative Function). In addition to being commutative and associative, our merge function usually reduces the entropy of the probabilistic database, a feature that makes it ideal for ER using entropy reduction.

Let us explain how CAF merges x-tuples. Suppose $W$ is the possible worlds set of two x-tuples. Let $W_m \subseteq W$, called mergeable possible worlds, be the set of possible worlds which contain mergeable tuples from the two x-tuples. Let $W_c \subseteq W$, called contradictory possible worlds, be the set of possible worlds that contain tuples that are not mergeable. For merging the two x-tuples, our merge function should preform the following steps:

1. Merging tuples in $W_m$ possible worlds, and aggregating the probabilities of the worlds that contain the same merge result.

2. Normalizing the tuples’ probabilities by a normalization factor $k$.

Any merge function that proceeds the above steps is commutative, but not necessarily associative. In fact, the value of the normalization factor greatly affects the associativity of the merge function. In CAF, we set the normalization factor equal to $1 - p_\Theta$, where $p_\Theta$ is the aggregated probability of the contradictory possible worlds, i.e. $W_c$. As shown in [79], the only normalization factor for which the merge function is associative is $1 - p_\Theta$. Indeed, this normalization factor normalizes the tuples’ probabilities so that their sum is equal to one. Let us now formally define the CAF merge function.

---

3This value is equal to the normalization factor of Dempster’s rule of combination [118].
5.3.3. DEFINITION. CAF merge function. Let \( x \) and \( x' \) be two \( x \)-tuples. Let 
\[
P_{\mathcal{D}} = \sum_{t \in x, t' \in x'} p(t)p(t'),
\]
where \( t \in x \) and \( t' \in x' \). Then, the result of merging \( x \) and \( x' \) by CAF is an \( x \)-tuple as follows:
\[
 x \odot x' = \{ t'' , p(t'') \} = \frac{\sum_{t \in x, t' \in x'} p(t)p(t')}{1 - P_{\mathcal{D}}}, t \in x, t' \in x', t'' \neq \emptyset
\]

For illustration, consider two \( x \)-tuples \( x_1 \) and \( x_2 \) in Figure 5.2(a). To compute \( x_1 \odot x_2 \), CAF merges every alternative of \( x_1 \) with every alternative of \( x_2 \). The resulted tuples are shown in Figure 5.2(b), where each cell in row \( t_i \) and column \( t_j \) shows the result of merging \( t_i \) with \( t_j \) and its probability of existence, which is equal to \( p(t_i)p(t_j) \). The corresponding attribute values of tuples \( t_4, t_5, t_2 \odot t_3, t_2 \odot t_4 \), and \( t_2 \odot t_5 \) are equal but non-identical, thus CAF considers these tuples as one tuple, say \( t' \), and adds their probabilities together. CAF keeps all distinct tuples and divides their probabilities by \( 1 - P_{\mathcal{D}} \), which is equal to 0.72. The resulted \( x \)-tuple is shown in Figure 5.2(c).

<table>
<thead>
<tr>
<th>( x )-tuple</th>
<th>( t )</th>
<th>name</th>
<th>phone</th>
<th>address</th>
<th>( p(t) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_1 )</td>
<td>( t_1 )</td>
<td>Thomas Michaelis</td>
<td>5256661</td>
<td>21 College Blvd.</td>
<td>0.4</td>
</tr>
<tr>
<td>( x_1 )</td>
<td>( t_2 )</td>
<td>Thomas Michaelis</td>
<td></td>
<td></td>
<td>0.5</td>
</tr>
<tr>
<td>( x_2 )</td>
<td>( t_3 )</td>
<td>T. Michaelis</td>
<td>5256661</td>
<td></td>
<td>0.3</td>
</tr>
<tr>
<td>( x_2 )</td>
<td>( t_4 )</td>
<td>T. Michaelis</td>
<td>5256661</td>
<td>45, Main street</td>
<td>0.2</td>
</tr>
<tr>
<td>( x_2 )</td>
<td>( t_5 )</td>
<td>Thomas Michaelis</td>
<td>5256661</td>
<td>45, Main St.</td>
<td>0.2</td>
</tr>
</tbody>
</table>

(a)

<table>
<thead>
<tr>
<th></th>
<th>( t_1 )</th>
<th>( t_2 )</th>
<th>( t_\perp )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_3 )</td>
<td>( t_2 \odot t_3 = t' ), 0.15</td>
<td>( t_2 \odot t_3 = t', 0.15 )</td>
<td>( t_1 \odot t_3 = t_3 ), 0.03</td>
</tr>
<tr>
<td>( t_4 )</td>
<td>( t_2 \odot t_4 = t' ), 0.1</td>
<td>( t_1 \odot t_4 = t_4 ), 0.02</td>
<td></td>
</tr>
<tr>
<td>( t_5 )</td>
<td>( t_2 \odot t_5 = t_2 ), 0.1</td>
<td>( t_1 \odot t_5 = t_5 ), 0.02</td>
<td></td>
</tr>
<tr>
<td>( t_\perp )</td>
<td>( t_2 \odot t_\perp = t_2 ), 0.15</td>
<td>( t_1 \odot t_\perp = t_\perp ), 0.03</td>
<td></td>
</tr>
</tbody>
</table>

(b)

<table>
<thead>
<tr>
<th>( x )-tuple</th>
<th>( t )</th>
<th>( p(t) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_1 \odot x_2 )</td>
<td>( t_1 )</td>
<td>0.12/0.72 = 0.17</td>
</tr>
<tr>
<td>( x_1 \odot x_2 )</td>
<td>( t_2 )</td>
<td>0.15/0.72 = 0.21</td>
</tr>
<tr>
<td>( x_1 \odot x_2 )</td>
<td>( t_3 )</td>
<td>0.03/0.72 = 0.04</td>
</tr>
<tr>
<td>( x_1 \odot x_2 )</td>
<td>( t_\perp )</td>
<td>(0.15 + 0.1 + 0.1 + 0.02 + 0.02)/0.72 = 0.54</td>
</tr>
</tbody>
</table>

(c)

Figure 5.2: a) Example of two \( x \)-tuples and their alternatives, b) CAF’s approach in merging the two \( x \)-tuples, c) The result of merging by CAF
5.3. **ERPD Using Entropy**

Figure 5.3: Entity reduction for two x-tuples $x = \{t\}$ and $x' = \{t'\}$ when $t$ and $t'$ are: a) Non-mergeable, b) Mergeable type 1, c) Mergeable type 2, d) Mergeable type 3. Blue, black, and red colors represent entropy reductions with positive, zero, and negative values respectively.

**Entropy Reduction Property**

As mentioned previously, an interesting property of the CAF merge function is that it usually reduces entropy. More precisely, let $x$ and $x'$ be two x-tuples, then, usually we observe that:

$$
\text{entropy-reduction-value} = \text{entropy}(x) + \text{entropy}(x') - \text{entropy}(x \odot x') > 0
$$

(5.6)

We call this property, entropy-reduction property of CAF.

To see the entropy-reduction property of CAF, consider a simple case of merging two x-tuples, each with only one alternative, i.e. x-tuples $x = \{t\}$ and $x' = \{t'\}$. We compute the entropy reduction value for a number of points representing $(p(t), p(t'))$ for each of the four merge possibilities of $t$ and $t'$ (i.e. non-mergeable, and the three mergeable types). The results are shown in Figure 5.3, where the points are colored depending on their entropy reduction value, i.e. blue, black, and red for positive, zero, and negative, respectively. This figure shows that merging the x-tuples using CAF may increase entropy only when the tuples are non-mergeable and both have high existence probabilities (i.e. the red area in Figure 5.3(a)). This shows that in the case of single alternative x-tuples, usually CAF reduces entropy. Although visualizing the entropy-reduction property for x-tuples with more than one alternative is not possible, we observe this behavior in our experiments.

The entropy-reduction property of CAF makes it appropriate for being used in our entity resolution algorithm which we describe in the next subsection.
5.3.3 ME Algorithm

Our entity resolution algorithm, called ME (i.e. Minimized Entropy), greedily merges x-tuples to generate a database whose entropy is close to the minimal entropy. ME gets a probabilistic database in the x-relation model as input and produces its cleaned version as output. Here, for ease of presentation, we assume that the input database is a single-alternative x-relation. This assumption is relaxed in Section 5.3.5.

Given a single-alternative x-relation $D$, ME works as follows:

1. Ignoring the tuples' probabilities, use an existing duplicate detection method to partition $D$ into a set $E$ of partitions such that each partition contains duplicate tuples which are believed to represent the same real-world entity.

2. For each partition $E_i$ in $E$ do steps 3 to 6.

3. Build the set $X_i$ of x-tuples by putting each tuple of $E_i$ in an x-tuple.

4. Let $x$ and $x'$ be two x-tuples in set $X_i$ for which the entity reduction value is maximum, say $r_{max}$.

5. If $r_{max} > 0$, then remove $x$ and $x'$ from $X_i$; add $x \odot x'$ to $X_i$; and goto step 4.

6. Add all of the x-tuples of $X_i$ to database $D^c$.

7. Return $D^c$.

In ME, we continuously search the two x-tuples whose merging provides the maximum entropy reduction with a positive value, remove them, and continue doing this until no merging with positive entropy reduction exists.

For merging the x-tuples, we use the CAF merge function. Notice that ME can work with any other merge function that is associative and commutative. However, using CAF makes the ME algorithm more efficient, because of its entropy-reduction property.

Let us now illustrate ME using an example.

**Example 3.** Consider database $D$ in Figure 5.4(a), where the schema of $D$ contains only one attribute $A$. Suppose we use a duplicate detection method that partitions $D$ into two partitions $E_1$ and $E_2$, where $E_1 = \{t_1, t_2, t_3, t_4, t_5\}$ and $E_2 = \{t_6, t_7\}$.

Beginning from partition $E_1$, ME builds set $X_1$ by putting each of the $E_1$'s tuples in one x-tuple, i.e. $X_1 = \{\{t_1 : 0.5\}, \{t_2 : 0.9\}, \{t_3 : 0.8\}, \{t_4 : 0.7\}, \{t_5 : 0.6\}\}$, where the existence probability of each tuple is shown in front of it. To find two x-tuples with maximum entropy reduction, ME computes the entropy reduction that results from merging every two x-tuples in $X_1$. Merging $\{t_4 : 0.7\}$ and $\{t_5 : 0.6\}$ provides the maximum entropy reduction, i.e. 1.32, thus, ME merges...
5.3. ERPD Using Entropy

<table>
<thead>
<tr>
<th>t</th>
<th>A</th>
<th>p(t)</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₁</td>
<td>a</td>
<td>0.5</td>
</tr>
<tr>
<td>t₂</td>
<td>a</td>
<td>0.9</td>
</tr>
<tr>
<td>t₃</td>
<td>b</td>
<td>0.8</td>
</tr>
<tr>
<td>t₄</td>
<td>c</td>
<td>0.7</td>
</tr>
<tr>
<td>t₅</td>
<td>c</td>
<td>0.6</td>
</tr>
<tr>
<td>t₆</td>
<td>d</td>
<td>0.1</td>
</tr>
<tr>
<td>t₇</td>
<td>e</td>
<td>0.3</td>
</tr>
</tbody>
</table>

(a) Database $\mathcal{D}$

<table>
<thead>
<tr>
<th>x-tuple</th>
<th>t</th>
<th>A</th>
<th>p(t)</th>
</tr>
</thead>
<tbody>
<tr>
<td>x₁</td>
<td>t₁₂</td>
<td>a</td>
<td>0.79</td>
</tr>
<tr>
<td>x₂</td>
<td>t₄₅</td>
<td>c</td>
<td>0.88</td>
</tr>
<tr>
<td>x₃</td>
<td>t₆</td>
<td>d</td>
<td>0.07</td>
</tr>
<tr>
<td></td>
<td>t₇</td>
<td>e</td>
<td>0.28</td>
</tr>
</tbody>
</table>

(b) Cleaned database $\mathcal{D}^c$

Figure 5.4: a) Probabilistic database $\mathcal{D}$, b) The cleaned database resulted from applying the ME algorithm to $\mathcal{D}$

them into x-tuple $\{t_{4,5} : 0.88\}$ and updates set $X_1$, i.e. $X_1 = \{{t_1 : 0.5}, \{t_2 : 0.9\}, \{t_3 : 0.8\}, \{t_{4,5} : 0.88\}\}$. Then, ME updates entropy reduction values by considering the merge of newly added x-tuple with other x-tuples. Merging $\{t_1 : 0.5\}$ and $\{t_2 : 0.9\}$ provides the maximum entropy reduction, i.e. 1.18, thus, ME merges them into x-tuple $\{t_{1,2} : 0.95\}$ and updates set $X_1$, i.e. $X_1 = \{{t_{1,2} : 0.95\}, \{t_3 : 0.8\}, \{t_{4,5} : 0.88\}\}$. After updating the entropy reduction values, ME finds that merging $\{t_{1,2} : 0.95\}$ and $\{t_3 : 0.8\}$ provides the maximum entropy reduction, i.e. 0.12, thus, ME merges them into x-tuple $\{t_{1,2} : 0.79, t_3 : 0.16\}$ and updates set $X_1$, i.e. $X_1 = \{{t_{1,2} : 0.79, t_3 : 0.16\}, \{t_{4,5} : 0.88\}\}$. At this stage, the maximum entropy reduction is equal to $-0.05$, thus, ME adds the x-tuples in set $X_1$ to the cleaned database, i.e. $\mathcal{D}^c$.

Partition $E_2$ contains only two tuples, thus $X_2 = \{{t_6 : 0.1}, \{t_7 : 0.3\}\}$. The entropy reduction which is resulted from merging the only two x-tuples in $X_2$ is equal to 0.16. As a result, ME merges the two x-tuples in $X_2$ into x-tuple $\{t_6 : 0.07, t_7 : 0.28\}$ and adds it to the cleaned database.

The cleaned database is shown in Figure 5.4(b). The entropy of $\mathcal{D}$ is equal to 5.39 and that of $\mathcal{D}^c$ is equal to 2.62, meaning that ME has reduced the entropy by a factor of 2. In this example, ME obtains the database with minimum entropy.

5.3.4 Time Complexity

In this section, we analyze the time complexity of the ME algorithm. Since the time complexity of the duplicate detection step of ME depends on the approach used, we do not consider this step in our analysis.

Let us analyze the time that ME spends for each partition. Let $d$ be the number of tuples in the partition. In the first stage, ME computes the entropy reduction value for all $\frac{1}{2}d(d-1)$ single-alternative x-tuple pairs in the partition and
it computes each entropy reduction value in $O(1)$ time. For efficient management of entropy reduction values, ME can use a priority queue. Thus, the first stage of ME takes $O(d^2 \log d)$ time. In each of the next stages, ME updates the priority queue with the entropy reduction values that result from merging the newly generated x-tuple with the existing x-tuples. In general, at stage $i$, ME computes $d - i - 1$ entropy reduction values. In worst case, all x-tuples have the same number of alternatives, i.e. $d/(d - i)$, thus, computing each entropy reduction value takes $O(d^2/(d - i)^2)$ time. Also, inserting each computed entropy reduction value in the priority queue is of $O(d \log d)$. In worst case, ME merges all x-tuples into one x-tuple. Thus, the time that ME spends on the whole stages, including the first stage, can be computed as follows:

$$T(d) = O(d^2 \log d) + \sum_{i=1}^{d-2} \frac{d^2}{d - i} \log d = O(d^2 \log d) + d^2 \log d(H(d) - 1 - \frac{1}{d})$$

where $H(d)$ is the $d$-th harmonic number which is bounded by $\log d$. Thus, $T(d)$ is of $O(d^2(\log d)^2)$.

Let $n$ be the number of tuples in the database and $m$ be the number of different real-world entities that the tuples represent. In the worst case, all tuples in the database represent only one real-world entity, i.e. $d = n$. Thus, the time complexity of ME in worst case is of $O(n^2(\log n)^2)$. On average, we can assume that partitions contain equal number of tuples, i.e. $d = n/m$. Thus, the average time complexity of ME is of $O\left(\frac{n^2}{m}(\log \frac{n}{m})^2\right)$.

### 5.3.5 Multi-Alternative X-relation Case

The algorithm for a multi-alternative x-relation only differs in the duplicate detection step. After partitioning the database, we may obtain partitions of duplicate tuples that do not respect the x-tuple correlations, meaning that the alternatives of an x-tuple may be put in different partitions. Since alternatives of an x-tuple are different beliefs about the same entity, it is obvious that they should be put in the same partition. Thus, to respect the x-tuple correlations, we have to combine all partitions that contain different alternatives of an x-tuple. To do so, we perform the following steps:

1. Let $\mathcal{D}$ be a multi-alternative x-relation. Let $E$ be a partitioning of $\mathcal{D}$ tuples where each partition contains duplicate tuples that are believed to represent the same real-world entity.

2. If $E$ contains two different partitions $E_i$ and $E_j$, where $\exists t \in E_i \land \exists t' \in E_j \mid t, t' \in x, x \in \mathcal{D}$, then remove $E_i$ and $E_j$ from $E$; add $E_i \cup E_j$ to $E$; and goto step 2.

3. Return $E$
5.4 Performance Evaluation

In this section, we evaluate the performance of our algorithm and its competitors through experimentation over both synthetic and real-world databases. We first describe our experimental setup. Then, we evaluate the performance of the algorithms in terms of execution time, entropy reduction, and the quality of query results.

5.4.1 Experimental Setup

To the best of our knowledge, there is no previous work dealing with the ERPD problem by entropy reduction. Thus, we compare our work with Koosh [94] which represents the best approach in the literature for dealing with the ERPD problem over x-relation databases. In contrast to ME however, Koosh assumes that all duplicate tuples are mergeable and uses a generic merge function that merges two duplicate tuples into a merged tuple. In order to compare this proposal with ME, we use two different merge functions to create two versions of Koosh as follows:

- Koosh-CAF: instead of merging all duplicate tuples into one tuple (as does Koosh), we use CAF to merge them into one x-tuple.
- Koosh-\(\times\): we use the product merge function, where two tuples are merged into a tuple whose probability is the product of the original tuples. If the two tuples have conflicting attribute values (i.e. non-mergeable), we put the attribute value of the tuple with higher probability in the merged tuple.

We implemented ME, Koosh-CAF, and Koosh-\(\times\) in Java, and tested them over both synthetic and real-world databases, thus covering all practical cases. Since these algorithms are not much different over single-alternative and multi-alternative x-tuples, for simplicity we test them over single-alternative databases.

We use a couple of parameters to control the characteristics of the synthetic databases in the experiments. The number of tuples in the database is denoted by \(n\). The average number of duplicate tuples, which represent one entity, is denoted by \(d_e\) (i.e. entity-degree). The percentage of mergeable duplicate tuples is denoted by \(m_p\).

We implemented two different scenarios for generating the synthetic databases. In the first scenario, we generate a database, denoted by SDB, without any specific semantics and use it to evaluate the scalability of our algorithm and also the amount of entropy reduction which our algorithm achieves. In the SDB database, each tuple is represented by a vector \((eid, D, p)\), where \(eid\) is a positive integer representing the entity which the tuple represents; \(D\) represents the attribute of the tuple; and \(p\) is a probability that the tuple belongs to the database. To generate the SDB database, we repeatedly generate the duplicate tuples representing an entity as follows: We first randomly generate a number, say \(n_e\), uniformly
Table 5.1: Real-world databases used in our tests

<table>
<thead>
<tr>
<th>name</th>
<th>attributes</th>
<th>tuples</th>
<th>entities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cora</td>
<td>12</td>
<td>1295</td>
<td>117</td>
</tr>
<tr>
<td>Restaurant</td>
<td>4</td>
<td>864</td>
<td>751</td>
</tr>
</tbody>
</table>

distributed in range \([1..2d_e - 1]\). Then, we generate \(n_e\) tuples, all having the same \(eid\), with the meaning that they represent the same entity. To implement the scenario where \(m_p\) percent of duplicate tuples are mergeable, we set the \(D\) attribute of \(m_p \times n_e\) tuples of the \(n_e\) generated tuples to the same value (e.g. zero) meaning that they are mergeable, and we also set the \(D\) attribute of the other tuples to non-equal numbers, meaning that they are non-mergeable. We generate \(p\) attributes of each tuple as a random number in range \((0..1]\). We use the following distributions for generating \(p\): 1) normal with 0.1, 0.5, or 0.9 as mean and 0.25 as standard deviation and 2) uniform. We repeat this process to generate other entities in the database.

In the second scenario, we generate a database, denoted by \(Objects\), which stores the distance of the astronomical objects to the Earth. We use the \(Objects\) database to evaluate the quality of the query results before and after applying our algorithm to the database. \(Objects\) database is resulted from aggregating several observatories’ databases, thus each astronomical object is represented by a number of tuples, i.e., duplicate tuples, each of which has come form one observatory’s database. As in the SDB database, each tuple in the \(Objects\) database is represented by a vector \((eid, D, p)\) but with different semantics for \(D\) and \(p\) attributes. The \(D\) attribute denotes the measured distance between the astronomical object and the Earth in light-year. Since error is inevitable in scientific measurements, we assume that \(D = dis + (error \times dis)\) where \(dis\) is the actual distance and \(error\) is the percentage of error introduced in measuring the distance. Notice that \(error\) is a real number in \((-1..+1]\). The \(p\) attribute denotes how much the \(D\) attribute is close to reality (i.e. \(dis\) value). To reflect this assumption, \(p\) is set to \(1 - |error|\). For instance if \(dis = 100\) and \(D = 120\), then \(p\) is set to 0.8.

We generate the probabilistic \(Objects\) database and its deterministic version, denoted by \(Objects_c\), as follows: We first generate \(dis\) as a random number uniformly distributed in range \([1..100]\) and add the vector \(t = (eid, dis, 1.0)\) to \(Objects_c\) database. Then, we generate the random number \(n_e\) that is uniformly distributed in range \([1..2d_e - 1]\). We then generate \(n_e\) tuples all having the same \(eid\) as tuple \(t\) meaning that they are all duplicate tuples of \(t\). We use a normal distribution with 0 as mean and 0.5 as standard deviation to generate \(error\) in each tuple and use this number to compute \(D\) and \(p\) attributes of the tuple as we explained above. Then, we add generated tuples to the \(Objects\) database. We repeat this process until generating \(n\) tuples for the \(Objects\) database.
5.4. Performance Evaluation

Table 5.2: Default setting of experimental parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$: number of tuples in the database</td>
<td>200,000</td>
</tr>
<tr>
<td>$d_e$: average number of duplicate tuples per entity</td>
<td>10</td>
</tr>
<tr>
<td>$m_p$: percentage of mergeable duplicate tuples</td>
<td>20%</td>
</tr>
</tbody>
</table>

We also evaluate our algorithm on two real-world databases. Table 5.1 lists the real-world databases that we use in our experiments. The Cora database [92] includes bibliographical information about scientific publications with 12 different attributes. This database includes 1295 tuples that describe 117 distinct publications (i.e. on average 11.1 duplicate tuples per publication). The Restaurant database [124] includes the name, address, city, and the type attributes of restaurants. This database includes 894 tuples about 751 distinct restaurants (i.e. 1.2 duplicate tuples per restaurant).

These two databases have been frequently used for duplicate detection tasks in related work, e.g. [92, 124, 12, 15, 27, 72, 60]. In order to convert these databases to their probabilistic version, we use the same probability distributions as in the synthetic data. We will show that the probability distributions, which we use for converting the real-world database to its probabilistic version, do not have a significant effect on the result of our algorithm.

The default settings of the experimental parameters are listed in Table 5.2. By default, we set the number of tuples in the database to 200,000, and the average number of duplicate tuples per entity to 10. When $m_p$ is not used as a varying parameter, we set it to 20%, which is relatively small, in order to evaluate our algorithm in a context that the majority of the tuples cannot be merged.

In order to evaluate the performance of our approach, we measure the following metrics.

1) **Execution time.** The execution time is the time elapsed from the initiation to the completion of the algorithm.

2) **Entropy.** This metric measures the uncertainty degree of the database. We compare the entropy of the given input database to the entropy of the cleaned (i.e. deduplicated) database, which is produced by applying our algorithm to the input database.

3) **Query-result-error.** This metric measures the percentage of error in the query results compared to the ground truth, i.e. the query result over the deterministic database. Let us formally define this metric. Let $\mathcal{D}$ be a probabilistic database and $\mathcal{D}_c$ be its deterministic version. Let $Q$ be a COUNT query over $\mathcal{D}$’s schema. Let $Q(\mathcal{D}_c)$ be the result of $Q$ over the database $\mathcal{D}_c$ and $Q(\mathcal{D})$ be the expected value of the result of $Q$ over the probabilistic database $\mathcal{D}$. We define
the query-result-error as

\[
\text{query-result-error}(Q, \mathcal{D}) = \frac{Q(\mathcal{D}_c) - Q(\mathcal{D})}{Q(\mathcal{D}_c)} \times 100.
\]

In our experiments, we use the probabilistic database Objects and its deterministic version (i.e. Objects_c) as the test databases. In order to evaluate the effect of our algorithm on the quality of the query results, we use a COUNT query Q to compare the query-result-error(Q, Objects) with the query-result-error(Q, cleaned-Objects), where cleaned-Objects is the database produced by applying our algorithm to the Objects database.

We conducted our experiments on a windows XP machine with Intel Core i3-2100 CPU (i.e. 3MB Cache and 3.10 GHz) and 4GB memory.

![Figure 5.5: Query-result-error vs. n](image)

![Figure 5.6: Query-result-error vs. d_e](image)

### 5.4.2 Performance Results

In this section, we report the result of our experiments. We first show the error bound of our algorithm in approximating the entropy minimized clean database, i.e. \( \mathcal{D}_{\text{min}}^c \).

To study the error bound of our algorithm, we consider a database of \( d \) duplicate tuples, and enumerate different partitionings of this database to compute its \( \mathcal{D}_{\text{min}}^c \). Since the number of different partitionings increases exponentially with \( d \), we can only increase \( d \) up to 12. In our experiments, we set \( m_p \) to 20%, and use databases with different probability distributions. To obtain more stable results, we repeat each experiment 10 times and take the average of the results. We observe that the percentage of error is less than 0.02%, meaning that the error of our algorithm is quite low. Although we cannot compute \( \mathcal{D}_{\text{min}}^c \) for larger values of \( d \), we observe that the entropy of the database resulted from our algorithm for
5.4. Performance Evaluation

Each entity (i.e. each partition of duplicate tuples) is quite close to zero, i.e. the minimum entropy.

Evaluating the Quality of the Query Results

We compare the quality of the query results over Objects and the databases which are produced by applying the algorithms Koosh-CAF, Koosh-\(\hat{\times}\), and ME to the Objects database (hereafter called Koosh-CAF, Koosh-\(\hat{\times}\), and ME databases for short), while varying the number of tuples in the database (i.e. \(n\)) and the average number of duplicate tuples per entity (i.e. \(d_e\)). We use the query-result-error metric for measuring the quality of the query results, and the following query as the test query: “return the number of astronomical objects whose distance from the Earth is greater than 50 light-years”.

With \(n\) increasing up to 2,000,000, \(d_e\) set to 10, and \(m_p\) set to 0, Figure 5.5 shows the query-result-error in the query results over the Objects and databases produced by the three algorithms. We observe that the Koosh-CAF and Koosh-\(\hat{\times}\) databases have the same query-result-error as the Objects database, but the query-result-error of the ME database is drastically less than that of the Objects (i.e. about 95%). The reason is that ME combines duplicate tuples, thus yielding a more accurate estimation for a COUNT query. However, Koosh-CAF and Koosh-\(\hat{\times}\) often keep duplicate tuples in the database without even establishing mutual exclusion relation between them, thus adversely affecting the estimation for a COUNT query.

Figure 5.6 shows how the query-result-error in the query results over the Objects, Koosh-CAF, Koosh-\(\times\), and ME databases increases with \(d_e\) increasing up to 100, \(n\) set to 200,000, and \(m_p\) set to 0. For the same reason as above, we observe that ME drastically reduces the query-result-error in the query results,
while Koosh-CAF and Koosh-$\hat{x}$ do not change it. Figures 5.5 and 5.6 also show that in ME, while the reduction in query-result-error is almost constant with $n$, it is linear with $d_e$. The reason is that the error in the result of a COUNT query over a probabilistic database is directly affected by the sum of the probabilities of the duplicate tuples in each entity, which is linear with $d_e$; as a result, increasing $d_e$ has a linear effect on the query-result-error over the Objects database. In the ME database, we combine all duplicate tuples of an entity to a number of $x$-tuples. In each $x$-tuple, the sum of probabilities of the tuples is less than 1 and the number of $x$-tuples in the ME database is almost independent from $d_e$, thus query-result-error over the ME database is almost independent from $d_e$. Thus, the reduction in query-result-error is linear in $d_e$.

**Effect of the Number of Tuples**

We evaluated the three algorithms over the SDB database with different distributions of $p$ while varying the number of tuples in the database, i.e. $n$.

With the number of tuples increasing up to 2,000,000 and the other parameters set as in Table 5.2, Figure 5.7 shows the execution time of ME. The execution time is almost linear in $n$ for all distributions. Not surprisingly, increasing the mean of the normal distribution, decreases the execution time. This observation shows that our algorithm stops sooner when the database contains more confident tuples. The reason is that combining high confidence tuples increases entropy, thus our algorithm stops sooner when the database contains more confident tuples.

![Figure 5.9: Execution time of ME vs. $d_e$ over normal and uniform databases](image)

![Figure 5.10: Entropy vs. $d_e$ over normal database with mean = 0.5](image)

With the number of tuples increasing up to 2,000,000 and the other parameters set as in Table 5.2, Figure 5.8 compares the entropy of the SDB database with that of the databases produced by Koosh-CAF, Koosh-$\hat{x}$, and ME algorithms for normal SDB database with mean = 0.5. The other tested normal and uniform
distributions also show the same trend. This figure shows that while entropy reduction in Koosh-CAF and Koosh-× is not significant, ME significantly reduces entropy, i.e. about 73%. We also observe that the entropy of the databases are linear in \( n \) in all distributions. However, different distributions do affect the coefficient in the linear relation between the entropy and \( n \): a very low or high mean value for \( p \) (e.g. 0.1 and 0.9) decreases it, but a value close to 0.5 increases it.

**Effect of the Average Number of Duplicate Tuples per Entity**

We studied the performance of the three algorithms over the SDB database with different distributions of \( p \) while varying the average number of duplicate tuples per entity, i.e. \( d_e \).

With \( d_e \) increasing up to 100 and the other parameters set as in Table 5.2, Figure 5.9 shows the results measuring the execution time of ME. Obviously, the execution time increases with \( d_e \) for all distributions because when the number of duplicate tuples per entity increases, more tuple pairs are checked to be combined. However, the increase is smooth so that ME is quite scalable with respect to \( d_e \). This figure also shows that the execution time decreases a little with the mean of \( p \) distribution. This is because combining tuples with high probability increases entropy and thus is avoided by ME.

![Figure 5.11: Execution time of ME vs. \( m_p \) over different databases](image1)

![Figure 5.12: Entropy vs. \( m_p \) over normal database with \( mean = 0.5 \)](image2)

With \( d_e \) increasing up to 100 and the other parameters set as in Table 5.2, Figure 5.10 compares the entropy of the SDB database with that of the databases produced by Koosh-CAF, Koosh-×, and ME algorithms for normal SDB database with \( mean = 0.5 \). The other tested normal and uniform distributions also show the same trend. This figure shows that while the entropy of the SDB, Koosh-CAF, and Koosh-× databases remains constant, the entropy of the ME database
Chapter 5. ERPD Using Entropy Reduction

decreases drastically with increasing $d_e$, and also the percentage of reduction increases with $d_e$. For instance, we observe that ME achieves 99% reduction in entropy over the normal SDB database with $mean = 0.5$ and $d_e = 100$, meaning that ME outperforms the other algorithms by a factor of 144. The reason is that since the number of tuples is fixed, the entropy of the SDB database remains constant. On the other hand, having a fixed number of tuples and increasing the number of duplicate tuples per entity means that more tuples have the chance to merge or form an $x$-tuple, thus reducing the number of possible worlds and the entropy of the resulted cleaned database. Thus, the percentage of reduction increases with $d_e$.

Effect of the Percentage of Mergeable Duplicate Tuples

We evaluated the performance of the three algorithms over the SDB database with different distributions of $p$ while varying the percentage of duplicate tuples that can be merged together, i.e. $m_p$.

With $m_p$ increasing up to 1 and the other parameters set as in Table 5.2, Figure 5.11 shows the execution times of ME. As we expect, the execution time decreases with $m_p$ for all distributions. The reason is that when $m_p$ increases, more tuples are merged together to form a merged tuple, thus reducing the number of tuple pairs that need to be checked for being combined by our algorithm. Again for the same reason, we observe that the execution time decreases a little with the mean of $p$ distribution.

With $m_p$ increasing up to 1 and the other parameters set as in Table 5.2, Figure 5.12 compares the entropy of the SDB database with that of the databases produced by Koosh-CAF, Koosh-$\times$, and ME algorithms for normal SDB database with $mean = 0.5$. We observe the same trend in other tested normal and uniform

![Figure 5.13: Result of applying the algorithms to the Cora database](image1)

![Figure 5.14: Result of applying the algorithms to the Restaurant database](image2)
5.5 Analysis against related Work

Beyond the work on entity resolution presented in Chapter 2, we identify the following areas of related work.

In the database literature, entropy has been used as a quality metric for query results over a probabilistic database [43, 42]. In [43], entropy is used to measure the quality of the range queries, nearest neighbor queries, AVG and SUM queries over a probabilistic database which contains a probabilistic attribute, modeled using a probability density function over an interval. In [42], the authors define entropy as a quality metric for the query results over a probabilistic database, and
propose a data cleaning algorithm with the goal of optimizing the expected quality improvement of the query results under a limited budget. Although entropy is meant to act as a quality metric for any query, the proposal in [42] can efficiently approximate it only for a particular subclass of queries over an x-relation. Our work differs from these proposals however in the way that we use entropy as a quality metric for the probabilistic database, rather than only for query results over it, and we propose an efficient approach for entity resolution using entropy.

5.6 Conclusion

In this chapter, we considered the problem of Entity Resolution for Probabilistic Data (ERPD). Using entropy as a quality metric for a probabilistic database, we proposed an efficient technique for computing the entropy of a probabilistic database in the x-relation model. We proposed a merge function for merging x-tuples. Based on our merge function, we proposed ME, a polynomial time algorithm for dealing with the ERPD problem with the aim of minimizing the entropy of the cleaned database. Our experimentation results over both real-world and synthetic data show that ME significantly reduces the entropy of the tested databases, improves the quality of the query results over tested databases; and outperforms drastically the best existing algorithms in the literature. They show that ME can reduce the entropy of the tested probabilistic databases up to 99%, when compared to the best algorithms in the literature.