On entity resolution in probabilistic data
Ayat, S.N.

Citation for published version (APA):
Ayat, S. N. (2014). On entity resolution in probabilistic data

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: http://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.
Chapter 2

Background and Related Work

In this chapter, we provide background on uncertain data models, and review related work on entity resolution, which are closely related to this dissertation. Related work on a few other relevant areas appear later in their relevant chapters.

2.1 Uncertain Data Models

As discussed in Chapter 1, uncertain data are common in many real-life applications. Managing such data has been receiving much attention in many application domains during the past few years. We have been witnessing an increasing number of proposals dealing with different problems such as querying, indexing, and mining uncertain data. An integral ingredient of such proposals is the uncertain data model.

In this section, we first describe the possible worlds semantics, which is an important concept on which the uncertain data models are built. We then discuss the $\mathcal{x}$-relation data model [10], which is the model that we use throughout the thesis. For a brief survey of other uncertain data models, the interested reader is referred to Appendix A.

2.1.1 Possible Worlds Semantics

In the possible worlds semantics, an uncertain database represents a number of deterministic possible database instances each of which is called a possible world. Each possible world may be associated with a probability of existence. Throughout this thesis, we denote the set of possible worlds of an uncertain database $\mathcal{D}$ by $PW(\mathcal{D})$.

To illustrate, Figure 2.1(a) shows the uncertain database $\mathcal{D}$, which stores the name and price of five products, collected automatically from web using some schema matching techniques. Due to the uncertainty which arises in data extraction from the web, $\mathcal{D}$ is not certain about the prices of products $b$ and $d$, 

thus these values are shown with a probability distribution over a set of values. As a result, $D$ represents four possible worlds, each shown together with their probability of existence in Figure 2.1(b).

<table>
<thead>
<tr>
<th>p-name</th>
<th>price ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>a</td>
</tr>
<tr>
<td>$t_2$</td>
<td>b</td>
</tr>
<tr>
<td>$t_3$</td>
<td>c</td>
</tr>
<tr>
<td>$t_4$</td>
<td>d</td>
</tr>
<tr>
<td>$t_5$</td>
<td>e</td>
</tr>
</tbody>
</table>

(a)

<table>
<thead>
<tr>
<th>p-name</th>
<th>price</th>
<th>p-name</th>
<th>price</th>
<th>p-name</th>
<th>price</th>
<th>p-name</th>
<th>price</th>
<th>p-name</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>a</td>
<td>25</td>
<td>$t_1$</td>
<td>a</td>
<td>25</td>
<td>$t_1$</td>
<td>a</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>$t_2$</td>
<td>b</td>
<td>10</td>
<td>$t_2$</td>
<td>b</td>
<td>20</td>
<td>$t_2$</td>
<td>b</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>$t_3$</td>
<td>c</td>
<td>15</td>
<td>$t_3$</td>
<td>c</td>
<td>15</td>
<td>$t_3$</td>
<td>c</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>$t_4$</td>
<td>d</td>
<td>25</td>
<td>$t_4$</td>
<td>d</td>
<td>25</td>
<td>$t_4$</td>
<td>d</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>$t_5$</td>
<td>e</td>
<td>30</td>
<td>$t_5$</td>
<td>e</td>
<td>30</td>
<td>$t_5$</td>
<td>e</td>
<td>30</td>
<td></td>
</tr>
</tbody>
</table>

$w_1 : 0.08$ $w_2 : 0.32$ $w_3 : 0.12$ $w_4 : 0.48$

(b)

SELECT p-name FROM $D$
WHERE price < 20

(c)

<table>
<thead>
<tr>
<th>p-name</th>
<th>probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_2$</td>
<td>$0.08 + 0.32 = 0.4$</td>
</tr>
<tr>
<td>$t_3$</td>
<td>$0.08 + 0.32 + 0.12 + 0.48 = 1.0$</td>
</tr>
</tbody>
</table>

(d)

Figure 2.1: a) Uncertain database $D$, b) Possible worlds of $D$, c) Query $Q$, d) The result of evaluating $Q$ on $D$

A common approach in uncertain data management literature is to use the possible worlds semantics to extend the well-known concepts in deterministic data to their corresponding concepts in uncertain data. For instance, using the possible worlds semantics, the semantics of querying an uncertain database is defined as: 1) applying the query to each possible world, and 2) obtaining the probability of each result by summing up the probabilities of the possible worlds which contain that result. To illustrate, consider evaluating the query $Q$, shown in Figure 2.1(c), on the uncertain database $D$, shown in Figure 2.1(a). The result is shown in Figure 1.d, where attribute probability shows the probability of belonging each tuple to the query result. For example, the value of attribute probability for tuple $t_2$ is obtained by adding the probabilities of possible worlds $w_1$ and $w_2$, where $t_2$ is part of the query result.
In Chapter 3, we use the possible worlds semantics to define the semantics of entity resolution in uncertain data.

The number of possible worlds of an uncertain database can be exponential to the database size, making it impractical to represent an uncertain database with the set of its possible worlds. Uncertain data models are proposed to overcome with this problem by compactly representing a large number of possible worlds. For instance in Figure 2.1, the uncertain database in Figure 2.1(a) is the compact representation of the four possible worlds which are shown in Figure 2.1(b).

There are two important concepts related to the uncertain data models, i.e. completeness and closure. An uncertain data model that can represent any set of possible worlds is said to be complete. On the other hand, an uncertain data model is closed under a given operation if the result of applying that operation on any database in the model can also be represented by the model. Although complete models provide the maximum expressiveness, they can be more complex than needed by the application. In general, closure is more important than completeness, because if a model is sufficient for representing application’s data, and it is closed under the operations which is needed by the application, then its completeness is of no matter.

A number of uncertain data models have been proposed in the literature, each of which is suitable for modeling uncertainty in a particular application domain (see Appendix A for a brief survey). We next discuss the x-relation, which is the uncertain data model that we use in the thesis.

2.1.2 X-relation Data Model

X-relation [10] is a recently proposed uncertain data model that has been extensively used in the literature, e.g. [145, 11, 39, 141, 42, 52], for representing uncertainty in a variety of application domains such as sensor networks, scientific data management, and spatial databases. In this model, a probabilistic database \( D \) consists of a number of x-tuples. Each x-tuple consists of a number of tuples, called alternatives, each associated with a probability value showing its likelihood of occurrence. The sum of the probability values of the x-tuple’s alternatives is less than or equal to one. The occurrence of alternatives is mutually exclusive. The x-tuples within the database are disjoint and can occur independently of each other.

To illustrate, Figure 2.2 shows an example database \( D \) in the x-relation model, and its possible worlds. \( D \) 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 \( t_3 \). The x-relation’s assumptions, i.e. mutual exclusion of alternatives and independence of x-tuples, greatly simplify the computation of possible worlds’ probabilities. For instance, the probability of the possible world \( W_3 \), denoted by \( P(W_3) \), is computed as the joint probability of two independent probabilistic events: among the alternatives of \( x_1 \), \( t_2 \) occurs; and none of the alternatives of \( x_2 \) occur. Thus,
as shown in Figure 2.2(b), \( P(W_3) \) is equal to \( P(t_2) \times (1 - P(t_3)) = 0.24 \). Notice that the probability that none of the alternatives of an \( x \)-tuple, say \( x \), occur is equal to \( 1 - \sum_{e \in x} P(t) \).

In this thesis, we distinguish between two types of \( x \)-relations: single and multi alternative. In the single-alternative \( x \)-relation, each \( x \)-tuple consists of only one alternative, and in the multi-alternative \( x \)-relation, there could be more than one alternative for an \( x \)-tuple.

\[
\begin{array}{|c|c|c|}
\hline
x \text{-tuple} & t & P(t) \\
\hline
x_1 & t_1 & 0.6 \\
 & t_2 & 0.3 \\
\hline
x_2 & t_3 & 0.2 \\
\hline
\end{array}
\]

(a)

\[
\begin{array}{|c|c|c|}
\hline
w_i & w_i \text{ members} & P(w_i) \\
\hline
w_1 & \emptyset & (1 - P(t_1) - P(t_2)) \times (1 - P(t_3)) = 0.08 \\
\hline
w_2 & \{t_1\} & P(t_1) \times (1 - P(t_3)) = 0.48 \\
\hline
w_3 & \{t_2\} & P(t_2) \times (1 - P(t_3)) = 0.24 \\
\hline
w_4 & \{t_3\} & (1 - P(t_1) - P(t_2)) \times P(t_3) = 0.02 \\
\hline
w_5 & \{t_1, t_3\} & P(t_1) \times P(t_3) = 0.12 \\
\hline
w_6 & \{t_2, t_3\} & P(t_2) \times P(t_3) = 0.06 \\
\hline
\end{array}
\]

(b)

Figure 2.2: a) An example probabilistic database \( D \) in the \( x \)-relation model, b) Possible worlds of \( D \)

### 2.2 Entity Resolution

A typical entity resolution (ER) process usually includes the following three phases:

- **Data preparation**: deals with the heterogeneity in the data coming from different sources.

- **Matching**: finds the duplicate tuples, i.e. the tuples that refer to the same real-world entity.

- **Merging**: merges the duplicate tuples into a merged tuple.

The aim of data preparation phase is to provide a coherent view of the data that are gathered from different sources, which can possibly be heterogenous. The
2.2. Entity Resolution

data preparation phase is often categorized under the three tasks of Extraction, Transformation, and Loading, which are called ETL for short.

During the extraction task, data are extracted from a diverse set of data sources which may range from web tables to files in a specific scientific proprietary format to regular relational tables. The transformation task then deals with the heterogeneity in the extracted data at the schema and data levels. At the schema level, schema matching techniques are used to find the data items that have the same semantics but are described under different attributes (or a combination of attributes) in different sources. Consider, for instance, tel and phone which both refer to the phone number of a person; or address and the combination of number, street, and city which both refer to an individual’s address. At the data level, the transformation task deals with the heterogeneity problems as much as possible to provide a standardized view of the data. Some of such problems are as follows:

- Known typographical errors or variations: e.g. “behavior” and “behaviour”.
- Using different representation formats: e.g. mm/dd/yy and yy/mm/dd formats for dates.
- Using different naming conventions: e.g. “Andrew S. Tanenbaum” and “Tanenbaum, Andrew S.”.
- Using abbreviations and nicknames: e.g. “Oracle Corporation” and “Oracle Co.”.

The last step of ETL is loading in which the prepared data is loaded into the database. The efficient implementation of data preparation phase can greatly speed up the entity resolution process.

One may expect identical duplicate tuples as the result of the data preparation phase. However, in practice, the data preparation phase results in non-identical duplicate tuples, which are challenging to find. The matching phase of the ER process aims at finding such tuples.

The merging phase of the ER process merges the duplicate tuples, found by the matching phase, into a single tuple. This phase lets the database to gather all of its information about one entity, which is scattered among different tuples, in a single tuple. This not only improves the quality of data in the database but also may benefit the matching phase itself, as shown by Benjelloun et al. in the Swoosh proposal [25], where early merging of duplicate tuples is used to improve the performance of the matching phase.

In the merging phase, an important challenge is merging duplicate tuples with conflicting attribute values. We distinguish between two types of conflicts: the conflicts that can be resolved by the merge function (i.e. resolvable), and those that cannot be resolved by the merge function (i.e. non-resolvable). Resolvable
conflicts may occur in cases where the duplicate tuples agree on the value of an attribute but they use different naming conventions for representing the attribute, e.g. consider “VLDB” and “very large databases” as two conflicting attribute values. Resolvable conflicts may even occur in cases that the duplicate tuples disagree on the value of an attribute, but the merge function resolves the conflict by combining the conflicting values. For instance, consider two conflicting attribute values 30 and 35 representing the age of a person, and a merge function that resolves this conflict by taking the average of these values.

A variety of techniques, referred to by canonicalization techniques, are used to compute the attribute values of the merged tuple using those of the duplicate tuples. The used techniques differ greatly based on the attribute data type and the application context. For instance, a technique is to use the longest name among the names in duplicate tuples, e.g. choosing “Andrew Stuart Tanenbaum” among the names in set \{ “Andrew S. Tanenbaum”, “A. S. Tanenbaum”, “Andrew Stuart Tanenbaum” \}. Another technique is to majority voting for choosing the representative value among a number of inconsistent categorical values. For example, if 2 out 3 attribute values say that a person is retired, then he is considered as retired. Keeping all duplicate tuples’ values in a set-valued attribute is another canonicalization technique. The canonicalization techniques, however, may lose the correct information and decrease the quality of the data in the database.

A number of recently proposed ER approaches, e.g. [12, 75] do not merge the duplicate tuples, but instead they keep them in the database and associate with each of them a probability value indicating the likelihood that the tuple represents the real-world entity. The probabilistic query evaluation techniques are then used for answering the queries over the resulted probabilistic database.

In the rest of this section, we first describe the matching techniques. We then discuss methods for speeding up the ER process. Then, we discuss ER for probabilistic data.

### 2.2.1 Matching Methods

An large body of literature on ER has been devoted to ER’s matching phase. The matching proposals can be broadly divided into attribute and tuple matching techniques. In this section, we first provide an overview on matching approaches and the metrics used for evaluating their effectiveness. Then, we describe techniques for matching the individual attribute values, which provide the basis for matching the tuples. We then review the proposals that deal with the matching problem at the tuple level.

#### Overview

In matching phase, the decision that whether or not two tuples are duplicate tuples is called a match decision. We can distinguish between two types of matching
2.2. Entity Resolution

proposals, i.e. pairwise and collective. In pairwise matching, match decision for every two tuple is made independently from the other tuples, but match decisions are made jointly for a set of tuples in collective matching proposals. Matching proposals that use clustering algorithms, and those that use generative models, e.g. [28, 111], are examples of collective approaches.

A simple pairwise matching approach is to consider two tuples as matching if their similarity is higher than a certain threshold, where the similarity is measured using one of the metrics for measuring the similarity between tuples, some of which are described in Section 2.2.1. A variation of this technique uses two thresholds $\mu_1$ and $\mu_2$, where $\mu_1 < \mu_2$, to mark two tuples as matching if their similarity is higher than $\mu_2$; as possible-match if it is between $\mu_1$ and $\mu_2$; and as non-match, otherwise. The possible-matches then are manually examined by human experts to determine whether they are matches or not.

An important shortcoming of pairwise matching is that it may result in inconsistent match decisions. Consider, for instance, the matching of tuples $t_1$, $t_2$, and $t_3$, where $t_1$ is matched with $t_2$, and $t_2$ is matched with $t_3$, but $t_1$ and $t_3$ are not matched together. Some pairwise matching proposals deal with this problem by adding the additional matches that result from transitive closure. The transitive closure operation, however, is very computationally expensive in some applications, as shown in [116].

To evaluate the performance of a matching proposal, we can consider the matching problem as a classification problem and use the precision, recall, and $F_1$ metrics, which are frequently used to evaluate the performance of the classification algorithms. To illustrate these metrics, let $M$ denote the set of tuple pairs that are matched together by the matching proposal, and $M_{true}$ is the set of tuple pairs that are true matches. Then, precision is the fraction of correctly matched tuple pairs to all pairs in $M$; recall is the fraction of correctly matched tuple pairs to all pairs in $M_{true}$; and $F_1$ is the harmonic mean of precision and recall, i.e.

$$P = \frac{|M \cap M_{true}|}{|M|}; \quad R = \frac{|M \cap M_{true}|}{|M_{true}|}; \quad \text{and} \quad F_1 = \frac{2 \cdot P \cdot R}{P + R}$$

where $P$ and $R$ respectively denote precision and recall. In collective matching proposals, we can also use the metrics which are specifically designed for evaluating the performance of clustering algorithms. Purity, normalized mutual information, and rand index are examples of such metrics.

**Attribute Matching**

Typographical variations in string data are the most common source of mismatch between duplicate tuples. As a result, the ER’s matching phase mostly relies on string matching techniques to find the duplicate tuples. A number of string matching techniques have been proposed in the literature, each of which is suitable
for particular types of string errors or variations. While errors might occur in non-string data as well, the research on matching non-string data is still in its infancy [61].

In this section, we review some of the string matching proposals in duplicate detection context. We also discuss the matching of non-string data. The matching methods are presented either in the form of a similarity or distance metric, either of which can easily be converted to the other one.

**Character-Based Metrics**

The character-based metrics are best suited for handling typographical errors in string data. We here review the following character-based metrics:

- Edit distance
- Jaro
- Q-grams

The edit distance between two strings $s_1$ and $s_2$ is the minimum number of edit operations that should be performed on $s_1$ to convert it into $s_2$, where the edit operations are as follows: 1) insert, i.e. inserting a character into the string, 2) delete, i.e. deleting a character from the string, and 3) replace, i.e. replacing a character with another character.

Depending on the cost which is associated to each edit operation, different variations of edit distance have been proposed. In its simplest form, the cost of all operations are equal to one. In such a case, the edit distance is usually called the Levenshtein distance [85]. The Levenshtein distance can also be defined using the following recursive formula:

$$L(s_1, s_2, i, j) = \min \begin{cases} L(s_1, s_2, i - 1, j - 1) & \text{if } s_1[i] = s_2[j] \\ L(s_1, s_2, i - 1, j) + 1 & \text{if } s_1[i] \text{ is replaced with } s_2[j] \\ L(s_1, s_2, i, j - 1) + 1 & \text{if } s_2[j] \text{ is inserted into } s_1 \\ L(s_1, s_2, i - 1, j) + 1 & \text{if } s_1[i] \text{ is deleted from } s_1 \end{cases}$$

where $L(s_1, s_2, i, j)$ is the Levenshtein distance between the first $i$ characters of string $s_1$ and the first $j$ characters of string $s_2$. This definition together with using dynamic programming methods enable us to efficiently compute the Levenshtein distance in $O(|s_1| \cdot |s_2|)$ time.

The Needleman-Wunsch distance [98] is another variation of edit distance which defines different costs for each character replacement, insert, and delete operations. Ristad et al. [113] proposed a method for automatically determining Needleman-Wunsch’s costs from a set of equivalent strings written in different variations.
Another variation of edit distance is the Smith-Waterman distance \[120\] in which the cost of edit operations at the beginning and the end of the string are more than that of in the middle of the string. This metric can better match the strings that use different titles for names or use the same title in different locations. Consider, for instance, the three strings “Prof. Andrew S. Tanenbaum”, “Andrew S. Tanenbaum, Prof.”, and “Mr. Andrew S. Tanenbaum”, which are matched using the Smith-Waterman metric within a short distance.

The \textit{affine gap} distance \[135\] is yet another variation of edit distance, which is suitable for matching the shortened or truncated strings, e.g. “A. S. Tanenbaum” and “Andrew Stuart Tanenbaum”. This metric extends the edit operations with two new operations \textit{open gap} and \textit{extend gap}, where the cost of extending the gap is usually less than that of opening the gap.

The \textit{Jaro} metric \[80\] is another effective character-based similarity metric, which is primarily used for matching short strings such as first and last names. The Jaro similarity value of two strings is defined based on the common characters of the two strings. To illustrate, let \(s_1 = a_1 \ldots a_m\) and \(s_2 = b_1 \ldots b_n\) be two given strings. Then, a character \(a_i\) in \(s_1\) is in common with \(s_2\) if there exists a character \(b_j\) in \(s_2\) such that \(a_i = b_j\) and \(|i - j| \leq \frac{1}{2}\min(|s_1|, |s_2|)\). Let \(s'_1 = a'_1 \ldots a'_k\) be the characters in \(s_1\) that are in common with \(s_2\), and similarly \(s'_2 = b'_1 \ldots b'_l\) be that of \(s_2\). Let the number of transpositions, say \(t\), be the number of positions in which \(s'_1[i] \neq s'_2[i]\). The Jaro similarity value then is defined as:

\[
\text{Jaro}(s_1, s_2) = \frac{1}{3} \left( \frac{|s'_1|}{|s_1|} + \frac{|s'_2|}{|s_2|} + \frac{|s'_1| - t/2}{|s'_1|} \right).
\]

The \textit{Jaro-Winkler} similarity metric \[139\] is a variant of the Jaro metric in which more emphasis is put on matching the first few characters of the two strings. This metric is defined as follows:

\[
\text{Jaro-Winkler}(s_1, s_2) = \text{Jaro}(s_1, s_2) + \frac{1}{10} \max (LCP, 4) \cdot (1 - \text{Jaro}(s_1, s_2)),
\]

where \(LCP\) is the length of the longest common prefix of \(s_1\) and \(s_2\).

Another family of metrics use groups of characters, denoted by \textit{q-grams} \[127, 126\], for matching the strings. Given a string \(s\), the q-grams of \(s\), denoted by \(q\text{-grams}(s)\), are substrings of length \(q\), which are obtained by moving a sliding window of size \(q\) over \(s\). For instance, the 2-grams of string “computer” are as follows: “co”, “om”, “mp”, “pu”, “ut”, “te”, “er”. The idea behind q-grams is that duplicate strings have a large number of q-grams in common. The metrics that use q-grams lie somewhere in between of character-based and token-based metrics, which we explain next, and the only reason that we categorize these metrics under character-based metrics is that, in contrast to token-based metrics, q-grams often have no meaning.

Q-grams are used in a variety of ways for measuring the similarity between strings. For instance, \textit{dice coefficient} metric \[34\] uses the number of shared q-grams between two strings and the total number of their q-grams for defining the
similarity value. More precisely, the dice coefficient of two strings $s_1$ and $s_2$ is defined as:

$$\text{dice}(s_1, s_2) = \frac{2 \cdot |\text{q-grams}(s_1) \cap \text{q-grams}(s_2)|}{|\text{q-grams}(s_1)| + |\text{q-grams}(s_2)|}.$$ 

As another example of metrics which use q-grams, one may represent the strings as vectors of q-grams, and uses the cosine of the angle between vectors for measuring the similarity between the strings. A major shortcoming of metrics that do not consider the location of q-grams in the strings is that they may assign a high similarity value to non-duplicate strings. Consider, for instance, two non-duplicate strings “xanex” and “nexan” with the same set of 2-grams, i.e. set \{“xa”, “an”, “ne”, “ex”\}, where measuring the similarity of these strings using dice coefficient results in the maximum similarity value, i.e. one. To deal with this problem, some approaches, e.g. [123, 68, 67], augment q-grams with their locations in the strings.

**Token-Based Metrics**

Character-based metrics mostly deal with typographical errors in duplicate strings. Another common source of difference in duplicate strings is the reordering of words in them, which occurs due to using different naming conventions, e.g. “Andrew S. Tanenbaum” and “Tanenbaum, Andrew S.”. This problem cannot be dealt with character-based metrics. As a result, token-based metrics have been proposed to deal with this problem. These metrics represent each string as a bag of tokens, i.e. words, and use a variety of techniques to compute their similarity. We here review some important token-based metrics.

A simple, yet effective, token-based metric is *Jaccard* similarity metric in which the similarity between two string $s_1$ and $s_2$ is defined as follows:

$$\text{Jaccard}(s_1, s_2) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|},$$

where $S_1$ and $S_2$ respectively are the set of words in $s_1$ and $s_2$.

An important metric, which is widely used in information retrieval community, is *TF-IDF* similarity metric [49]. TF-IDF represents each string $s$ as a vector $V = (v_1, \ldots, v_n)$ whose $i$th element, i.e. $v_i$, represents the $i$th word in the corpus, say $w_i$, and is equal to:

$$v_i = \log(tf(w_i) + 1). \log(idf(w_i))$$ \hspace{1cm} (2.1)$$

where $tf(w_i)$ is the number of times that word $w_i$ appears in string $s$, and $idf(w_i)$ is $\frac{n}{n_i}$, where $n$ is the total number of strings in the corpus, and $n_i$ is the number of strings in the corpus that contain word $w_i$. TF-IDF then normalizes each vector into unit vector by dividing it by its $l_2$ norm, i.e. its weight, and computes the dot product of two vectors, which is equal to the cosine of the angle between
them, as the similarity of the strings that the vectors represent. More precisely, the TF-IDF similarity between strings \( s \) and \( s' \) is defined as:

\[
\text{TF-IDF}(s, s') = \frac{\sum_{i=1}^{n} v_i \cdot v'_i}{\sqrt{\sum_{i=1}^{n} v_i^2 \cdot \sum_{i=1}^{n} v'_i^2}}.
\]

In the TF-IDF metric, the similarity is affected only by words that appear in both \( s \) and \( s' \), and the metric is in favor of the words that are rare in the corpus, and appear a large number of times either in \( s \) or \( s' \), meaning that having such words in the two strings results in higher similarity value between them. For instance, in a corpus of university names, matching rare words such as “Stanford” and “Amsterdam” are of more importance than matching frequent words such as “university” and “of”. The TF-IDF works can effectively match duplicate words having different ordering of words. Moreover, this metric is not sensitive to the introduction of frequent words in duplicate words. For instance, the similarity between strings “Andrew Tanenbaum” and “Tanenbaum, Andrew” is equal to one, and the similarity between string “Mr. Andrew Tanenbaum” and these strings is very close to one.

An important shortcoming of the TF-IDF metric is that it cannot capture the typographical errors in the input strings. For instance, the two duplicate strings “University of Amsterdam” and “Amsterdam University” are assigned a zero similarity value by the TF-IDF metric. To deal with this problem, Bilenko et al. [31] proposed the SoftTF-IDF metric. In the SoftTF-IDF metric, the similarity is affected by the words that appear in both strings and also the words in the two strings that are similar. To illustrate, let \( f \) be a similarity metric that works well on short strings, e.g. Jaro-Winkler metric, and \( \mu \) in range \((0..1]\) be a threshold. Let \( \text{similar}(s, s') \) be the set of words \( w \in s \) so that there exists a word \( w' \in s' \), where \( f(w, w') > \mu \), and for each word \( w \) in \( \text{similar}(s, s') \), let \( \text{sim}(w) = \max_{w' \in s'} f(w, w') \). Also, let us extend \( v_i \) in equation (2.1) to \( v_i(w, s) \) to clearly show its relevance to word \( w_i \) and string \( s \). Then, the similarity of \( s \) and \( s' \) according to SoftTF-IDF is defined as:

\[
\text{SoftTF-IDF}(s, s') = \frac{\sum_{w \in \text{similar}(s, s')} v(w, s) \cdot v(w, s') \cdot \text{sim}(w)}{\sqrt{\sum_{w \in \text{similar}(s, s')} v(w, s)^2} \cdot \sqrt{\sum_{w \in \text{similar}(s, s')} v(w, s')^2}}.
\]

Setting \( \mu \) to one in the above equation results in the same similarity values as the TF-IDF metric.

Using q-grams as tokens, instead of words, in the TF-IDF metric is another method that has been proposed to deal with the TF-IDF problem of typographical errors in the strings [69].
Metrics for Non-String Data

Non-string data ranges from simple data such as numbers and dates to complex data such as images, audios, and videos. While matching complex data is often the subject of a whole field of research, e.g. matching images in computer vision field, the proposed methods for matching simple non-string data are rather primitive. One common method for matching simple non-string data is to treat them as strings and use string matching techniques. Other matching methods are limited to simple techniques such as using maximum absolute or relative difference for numbers, and converting dates to days and using maximum absolute difference, e.g. within 20 days of each other.

Effectiveness of Different Metrics

As we discussed above, there exist a large number of attribute value matching metrics in literature, which confirms the fact that there is no single metric that fits all datasets and all types of errors in data. However, some metrics have been shown to perform better than the other metrics, on average, over different datasets.

As shown in [31, 50], Monge-Elkan [96, 97], which is a tuned version of affine gap metric, Jaro-Winkler, and SoftTF-IDF metrics have the best performance. The Monge-Elkan metric has the best average performance among character-based metrics. This method, however, does not scale well. Jaro-Winkler is the best performer for matching the names, and SoftTF-IDF has the best average performance among all metrics.

Tuple Matching

In this section, we present the methods that deal with the matching problem at the tuple level. These approaches can broadly be divided into three categories:

- **Distance-based**: approaches that use a generic distance (or similarity) metric to match tuples.
- **Machine learning based**: approaches that need training data to match tuples.
- **Rule-based**: approaches that rely on matching rules, which are specified with the involvement of human experts, to match tuples.

In general, learning-based and rule-based approaches outperform distance-based proposals in terms of accuracy. However, distance-based approaches are more practical since they neither need training data nor human expertise. In the rest of this section, we present tuple matching techniques in these categories in different subsections.
Distance-Based Approaches

The main component of distance-based approaches is a metric for measuring the distance (or similarity) between tuples. In this section, we discuss some of the approaches for implementing such metrics.

One approach is to treat each tuple as a long attribute, and use one of the attribute value matching metrics to match the tuples. This approach however ignores the tuple structure, and may yield incorrect matching results. For instance, in TF-IDF metric, a certain word may be rare in the values of one attribute, but common in the other one. Thus, considering all attributes together may result in the vectors that incorrectly reflect the rarity of words within individual attribute values.

To consider the tuple structure, an approach is to use the appropriate metrics to measure the similarity between individual attribute values, and then compute the weighted similarity as the similarity between tuples. The advantage of this approach is that we can adjust the relative importance of each attribute by varying its weight. However, the weights are challenging to compute and cannot be changed dynamically, but should be fixed a priori. To overcome these drawbacks, some approaches do not use fixed weights, but compute variable weights dynamically. For instance, FlexiTF-IDF [81], a variation of TF-IDF, normalizes the TF-IDF vector representation of the attribute values using a joint normalization factor to implement a dynamic weighting scheme. To illustrate, without loss of generality, let $t$ be a tuple on schema $(a, b)$, and vectors $V_a$ and $V_b$ be the TF-IDF representation of $t$'s attribute values. Instead of normalizing each vector by dividing it by its $l_2$ norm, as does the TF-IDF, the FlexiTF-IDF normalizes vectors $V_a$ and $V_b$ by dividing them by $\sqrt{\|V_a\|^2 + \|V_b\|^2}$, where $\|V_a\|$ and $\|V_b\|$ are the $l_2$ norms of $V_a$ and $V_b$, respectively.

In Chapter 3, we introduce CB, a distance-based similarity metric that needs neither statically specified nor dynamically computed attribute weights, but instead relies on the rarity of individual attribute values in the database.

Ranked-list-merging [71] is another proposal that deals with measuring the distance between tuples. The idea is that, to find the best matches for a tuple, we can rank database tuples based on their similarity with the given tuple based on only one attribute, where the similarity is measured by an appropriate similarity metric. Repeating this process for all attributes results in $m$ ranked lists of database tuples, where $m$ is the number of attributes. This proposal then aims at assigning final ranks to tuples to minimize the distance between the final assigned ranks and the ranks that are assigned by individual attributes, where the distance between different ranks is measured by the footrule distance [56]. The ranked-list-merging proposal deals with this problem by providing efficient solutions for identifying the top-k matching tuples.

An important problem of distance-based pairwise approaches is choosing the appropriate distance threshold. Chaudhuri et al. [38] proposed a method to deal
with this problem. They observed that a fixed threshold value for all entities, i.e. sets of duplicate tuples, is not the right choice, and different entities need different threshold values. They identified two properties of duplicate tuples, i.e. compact set, and sparse neighborhood properties. Intuitively, these properties say that duplicate tuples that represent the same entity are closer to each other than to the other tuples, and their local neighborhood is sparse. Chaudhuri et al. exploited these properties to propose a variable thresholding scheme which outperforms approaches that rely on a fixed threshold value for all entities.

Machine Learning-Based Approaches

Most of the learning-based approaches model the tuple matching problem as a classification problem in which the tuple pairs are assigned to one of the two match and non-match classes, denoted by $M$ and $N$, respectively. In a variation, a third possible-match class is also considered. Then, the tuple pairs that are assigned to the possible-match class are further investigated by human experts to be assigned to their real classes.

The learning-based approaches can be divided into three categories based on their required amount of training data, which is in the form of tuple pairs prelabeled as match or non-match. These categories include supervised, which needs a considerable amount of training data, semi-supervised, which uses only a few training data, and unsupervised, which works with no training data.

One of the early learning-based approaches is that of Fellegi and Sunter [62]. They adapted the Naive Bayes classification method for the tuple matching task. In their proposed approach, a comparison vector $V_\rho = (v_1, \ldots, v_m)$ is associated to each tuple pair $\rho = (t, t')$, where $0 \leq v_i \leq 1$ is the level of agreement between the $i$th attribute value of $t$ and $t'$, and the tuple pair's class label, denoted by $\text{class}(\rho)$, is computed as follows:

$$\text{class}(\rho) = \arg\max_{c \in \{M,N\}} P(c) \prod_i P(v_i | c)$$

The probabilities $P(c)$ and $P(v_i | c)$ in the above equation can be either computed using training data or estimated, as does Winkler [138] by using the expectation maximization method [54], when no training data is available.

A variety of supervised classification algorithms have been adapted for the tuple matching task, e.g. decision trees [46], support vector machines [30, 45], conditional random fields [73, 93, 57], and ensemble of classifiers [41]. The effectiveness of these approaches heavily depends on the training set with which the classifier is trained. Generating a suitable training set however is a challenging task because most of the tuple pairs are easily classifiable non-matches, and the close non-match tuples with subtle differences, which can train a highly accurate classifier, are very hard to find. To deal with this problem, some tuple matching approaches, e.g. [114, 124, 14, 24, 133, 91], use the active learning method [51].
The basic idea in active learning is that the learning algorithm actively participates in the task of choosing the subset of unlabeled data whose labeling and inclusion in the training set most likely improves the accuracy of the learning algorithm.

Another solution to the hard task of generating large amount of appropriate training data is to use semi-supervised or unsupervised learning approaches. For instance, Verykios et al. [131] proposed a tuple matching approach that uses only a few labeled data. This approach uses the attribute value matching techniques to compute a comparison vector for each tuple pair, and then the AutoClass [40] method is used to cluster the comparison vectors into a set of clusters. The basic assumption is that a cluster contains similar comparison vectors that hopefully belong to the same match or non-match class. Thus, by knowing the label of only a few vectors in a cluster, it is possible to determine the class label of all vectors in the cluster. Verykios et al. has shown that their approach works effectively using only a minimal number of labeled data.

There are learning-based tuple matching approaches that work in an unsupervised setting. Bhattacharya and Getoor [28] adapted the Latent Dirichlet allocation model [33] for the tuple matching task. In this approach, attribute values are modeled using latent variables which then are used to build a generative model which is used for matching the tuples by inference on groups of values that commonly occur together. Ravikumar and Cohen [111] also used latent variables to propose a generative model for the tuple matching task. In their proposed model, each element of the tuple pair’s comparison vector is modeled using a binary latent variable which shows whether the corresponding attributes of the two tuples match or not.

**Rule-Based Approaches**

The core of rule-based approaches is a set of rules that specify under which conditions two tuples match. The set of rules differs between datasets and often is determined by human experts with the help of a declarative language. As an example of such rules, consider the following rule, which matches persons in an employee database:

$$sim(t_1.name, t_2.name) \geq 0.9 \land t_1.byear = t_2.byear \implies Match(t_1, t_2),$$

where $sim$ is an attribute value matching metric, and $byear$ represents the person’s birth year.

Although it has been shown that rule-based approaches can effectively be used for the tuple matching task [76, 134, 88, 64], heavy reliance on human expertise makes them impractical for real-life databases. As a result, a combination of learning-based and rule-based approaches is used in practice. In this model, training data is used to learn a set of rules which then are manually validated and refined by human experts.
Usability of Matching Methods in ERPD

We now discuss the usability of different matching methods, presented in this section, in the approaches that we present in this thesis for dealing with the ER problem in probabilistic data (called ERPD).

Our approach for dealing with the identity resolution problem (see Definition 1.2.1) in probabilistic data, presented in Chapter 3, can use any distance-based tuple matching metric for computing the similarity between tuples. Our approach however differs for different types of similarity metrics based on whether the similarity between two tuples depends on the other tuples in the database or not. Most of the existing similarity metrics of the former type (called context-sensitive), i.e. where the similarity of two tuples depends on the other tuples in the database, perform poorly when used within our proposed approach for the identity resolution problem in probabilistic data. To overcome this shortcoming, we propose a new similarity metric in Section 3.4.3, called $CB$, with the following features:

- By working at the attribute level, rather than at the word or q-gram level, $CB$ significantly reduces the number of rather costly string comparison operations which thus makes it very efficient compared to other context-sensitive similarity metrics.

- In contrast to most of the tuple matching methods that work at the attribute level, $CB$ does not need the specification of weights for representing the relative importance of individual attributes.

Our approach for dealing with the deduplication problem (see Definition 1.2.2) in probabilistic data, presented in Chapter 5, is generic, meaning that it can work with any matching method for deterministic data, including the ones that are discussed in this section.

2.2.2 Scalability Issues

The ER process is prohibitively expensive even for medium-sized databases. Identifying duplicate tuples, using a pairwise matching technique, on a database of size $n$, is of $O(n^2)$, since each tuple needs to be compared against all other $n - 1$ tuples in the database. The situation is worse in collective approaches since they rely on learning based methods, which do not scale well, as we discussed in previous section. In this section, we review some techniques for increasing the efficiency of ER proposals, which thus make them scalable for real-life databases.

Blocking is one of the early techniques for increasing the efficiency of ER methods. The idea of blocking is to divide the database into a number of blocks and only compare tuple pairs that reside within the same block, assuming that tuples from different blocks are unlikely to match. For example, one may use the
2.2. Entity Resolution

city attribute to partition a database of restaurants to a number of blocks, with the reasonable assumption that restaurants from different cities do not match. Blocks may overlap with each other, and usually are computed using one or multiple simple blocking attributes, e.g. city in the above example.

Blocking reduces the quadratic time complexity of pairwise ER techniques to \( O(b \cdot n) \), where \( b \ll n \) is the average number of tuples in a block. This increased efficiency comes with the cost of possible decrease in the accuracy of the ER method. Such possible accuracy decrease occurs due to missing duplicate tuples that may fall into different blocks, which, for instance, happens due to noisy or null values in the blocking attributes of the tuples.

While traditional blocking techniques, e.g. [99, 137, 29, 95, 70] process each block separately, recently, Whang et al. [136] have proposed a blocking method in which the matching decisions are communicated between blocks. The idea behind this approach is that when two tuples match and are merged in a block, the newly created merged tuple may match with the tuples in other blocks. Thus, as shown by Whang et al., communicating the merged tuple to other blocks increases the accuracy of ER, and even may increase the efficiency by avoiding unnecessary comparisons in other blocks.

Another approach for increasing the efficiency of ER is the use of canopies [92]. The idea is to use a cheap distance metric to quickly divide the tuples into a number of overlapping clusters, called canopies, and then use an exact, and thus more expensive, distance metric to perform a pairwise comparison on all tuples that have at least one canopy in common. For example, one might use the proportion of common q-grams of two strings as a cheap distance metric for computing canopies, and the edit distance between the strings, with tuned cost for the edit operations, as the more expensive metric. It has been shown that canopies outperform the traditional blocking methods in terms of both efficiency and accuracy [23].

In order to increase the accuracy, Rastogi et al. [110] have proposed a method in which match decisions are communicated between canopies. They run an instance of the matcher separately on each canopy, and when a match decision is made in a canopy, they pass the decision to other canopies, and rerun the matcher on them using the new communicated match decision. This process continues until no new matches is found on each canopy. Rastogi et al. have shown that the time complexity of their approach is of \( O(k^2 \cdot f(k) \cdot c) \), where \( k \) is the maximum size of a canopy; \( f(k) \) is the spent time on a canopy of size \( k \); and \( c \) is the number of canopies. The important advantages of this approach are the followings: 1) it can use a broad class of existing matchers; and 2) it is easily parallelizable using the MapReduce framework.

Achieving scalability in dealing with the ERPD problem is more challenging since we have to deal with an exponential number of possible worlds of probabilistic data. Our proposed approaches for dealing with the ERPD problem however are scalable, as shown through computational complexity analysis, and experi-
mentation. Moreover, we believe that our CB similarity function, presented in Chapter 3, can act as a cheap similarity metric in the blocking methods that, as explained above, aim at improving the efficiency of ER in deterministic data. Elaborating on this idea however remains as a possible direction for future research (see Section 7.2).

2.2.3 Entity Resolution for Probabilistic Data

While the ER problem has been well studied in the literature for deterministic data, only a few proposals have addressed the problem of ER in probabilistic data (ERPD). In this section, we review these proposals.

Matching X-tuples

Panse et al. [103] have proposed a method for matching x-tuples in the x-relation probabilistic data model. In their proposed method, two x-tuples are matched if their expected similarity is above a certain specified threshold, where the expected similarity between two x-tuples is computed by combining the similarity and the probability of their alternatives. More precisely, the expected similarity between x-tuples $x$ and $x'$ is defined as follows:

$$\text{Sim}_{\text{exp}}(x, x') = \sum_{t \in x} \sum_{t' \in x'} \frac{p(t)}{p(x)} \cdot \frac{p(t')}{p(x')} \cdot \text{Sim}(t, t'),$$

where $p(x) = \sum_{t \in x} p(t)$, and similarly for $p(x')$.

By ignoring the probability distribution of possible worlds, which is the common drawback of expected values, this proposal may result in unreliable ranking of x-tuples.

Another drawback is that single-alternative and multi-alternative x-tuples are treated differently because probabilities are ignored completely for single-alternative x-tuples.

To overcome these shortcomings, we propose a new approach for matching tuples and x-tuples in the x-relation data model in Chapter 3.

Merging X-tuples

[102] is a proposal by Panse et al. that addresses the merging of x-tuples in an extended version of the x-relation model, where each x-tuple is associated with a probability value indicating its membership degree to the database. In order to merge two x-tuples, they assign a weight to each x-tuple, where sum of the weights is equal to one; multiply the probability of each tuple by the weight of the x-tuple to which it belongs; unify the tuples of the two x-tuples into the merged x-tuple; and combine the identical tuples by adding their probabilities.
As an example, consider merging two x-tuples $x$ and $x'$ shown in Figures 2.3(a) and 2.3(b), respectively. Assigning weights 0.6 and 0.4 respectively to $x$ and $x'$, the resulted merged x-tuple, denoted by $x \odot x'$, is shown in Figure 2.3(c). The membership probability of $x \odot x'$ is not shown in the Figure since it depends on the interpretation of relationship between source relations, to which $x$ and $x'$ belong, and the destination relation of the $x \odot x'$.

The important drawback of this proposal is that it does not merge tuples that are mergeable but not identical (i.e. tuples with null values), e.g. tuples $t_3$ and $t_4$ in Figure 2.3. Thus, the whole data that the database has about an entity, is not aggregated in one tuple, which thus adversely affects the quality of the query results over the database. Our merge function, proposed in Chapter 5, does not suffer from this shortcoming.

**Deduplication in Probabilistic Data**

Koosh [94] is a proposal for dealing with the ER problem, more specifically deduplication definition (see Definition 1.2.2), over the x-relation data model. Koosh takes a pairwise approach for matching the tuples using a generic match function.

---

1Two tuples are mergeable if their attribute values do not conflict with each other. Consider, for instance, two tuples $t_1 = ("A. S. Tanenbaum", \perp)$ and $t_2 = (\perp, "Vrije Universiteit")$. 

---

<table>
<thead>
<tr>
<th>x-tuple t</th>
<th>name</th>
<th>age</th>
<th>phone-no</th>
<th>p(t)</th>
<th>p(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>John</td>
<td>35</td>
<td>5256662</td>
<td>0.25</td>
<td>0.8</td>
</tr>
<tr>
<td>$t_2$</td>
<td>John</td>
<td>30</td>
<td>5256662</td>
<td>0.45</td>
<td></td>
</tr>
<tr>
<td>$t_3$</td>
<td>John</td>
<td>\perp</td>
<td>7895226</td>
<td>0.3</td>
<td></td>
</tr>
</tbody>
</table>

(a)

<table>
<thead>
<tr>
<th>x-tuple t</th>
<th>name</th>
<th>age</th>
<th>phone-no</th>
<th>p(t)</th>
<th>p($x'$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>John</td>
<td>35</td>
<td>5256662</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>$t_2$</td>
<td>John</td>
<td>30</td>
<td>5256662</td>
<td>0.35</td>
<td>0.5</td>
</tr>
<tr>
<td>$t_4$</td>
<td>John</td>
<td>32</td>
<td>\perp</td>
<td>0.25</td>
<td></td>
</tr>
</tbody>
</table>

(b)

<table>
<thead>
<tr>
<th>x $\odot$ x' t</th>
<th>name</th>
<th>age</th>
<th>phone-no</th>
<th>p(t)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>John</td>
<td>35</td>
<td>5256662</td>
<td>0.25 \times 0.8 + 0.4 \times 0.2 = 0.28</td>
</tr>
<tr>
<td>$t_2$</td>
<td>John</td>
<td>30</td>
<td>5256662</td>
<td>0.45 \times 0.8 + 0.35 \times 0.2 = 0.43</td>
</tr>
<tr>
<td>$t_3$</td>
<td>John</td>
<td>\perp</td>
<td>7895226</td>
<td>0.3 \times 0.8 = 0.24</td>
</tr>
<tr>
<td>$t_4$</td>
<td>John</td>
<td>32</td>
<td>\perp</td>
<td>0.25 \times 0.2 = 0.05</td>
</tr>
</tbody>
</table>

(c)
A generic *merge* function is then used to merge the matched tuples into a tuple. Koosh is built on the assumption that the confidence of match function in matching two tuples affects the probability of the resulted merged tuple. For instance, consider three tuples $t_1$, $t_2$, and $t_3$ all having equal probabilities, and suppose that the match function is 80% sure that $t_1$ and $t_2$ are a match, and 40% sure that $t_2$ and $t_3$ are a match. Then, the tuple resulted from merging $t_1$ and $t_2$ gets higher probability than that of merging $t_2$ and $t_3$.

The above assumption makes the order in which the tuples are merged important. As a result, it is required to consider different orders for merging the tuples, which thus makes the ERPD (Entity Resolution in Probabilistic Data) problem computationally more expensive than ER over deterministic data.

Besides dealing with the ERPD problem, where match and merge functions are generic, Koosh has a number of more efficient variations, which are applicable to match and merge functions with specific properties.

One of improvements is achieved by introducing the concept of *domination*. Tuple $t$ is said to be *dominated* by tuple $t'$, if $t'$ contains all of the attribute values of $t$, and its probability is equal or higher than that of $t$. The match and merge functions are said to have the domination property if the dominated tuples do not participate in the generation of non-dominated tuples. In such a case, Koosh removes dominated tuples as soon as they are generated in the ER process. Early removal of dominated tuples improves the efficiency of the ER process.

The *threshold* property is another property, which is used to improve the efficiency of Koosh. The threshold property holds if a tuple whose probability is less than $\mu$ cannot participate in the generation of a tuple whose probability is greater than $\mu$, where $\mu$ is a user-specified threshold. In the match and merge functions, for which the threshold property holds, the below-threshold tuples are removed as soon as they are generated in the ER process. This improves the efficiency of the ER process.

Koosh removes a duplicate tuple $t$ from the database only when there is another tuple that contains all data as $t$ and has a higher probability than $t$. This causes that Koosh often keeps the original duplicate tuples together with the merged tuple in the cleaned database without adjusting their probabilities and establishing a mutual exclusion relation between them. Thus, the amount of data quality improvement by Koosh may not be noticeable.

The second shortcoming is that Koosh requires that the outcome of the merge function be one tuple. In merging tuples with conflicting attribute values, this requirement makes the merge function to either discard one of the values or keep both values in the attribute. In the former case, the information is lost by the merge function, and in the latter case, matching the merged tuple with other tuples is hard to be done since the merged tuple represents several tuples not one.

The third shortcoming is that Koosh just deals with the ER problem in single-alternative x-relations, and does not consider multi-alternative x-relations.
2.2. Entity Resolution

Our proposal, presented in Chapter 5, deals with the deduplication problem in the x-relation data model, while overcoming the shortcomings of Koosh method.

2.2.4 Conclusion

In this chapter, we reviewed the main approaches for dealing with the ER problem both in deterministic and probabilistic data.

The only proposed approach in the literature [103] for matching x-tuples in the x-relation probabilistic data model, which is the model on which we focus in this thesis, is based on expected values which has two main drawbacks: 1) it may result in unreliable matching of x-tuples due to ignoring the probability distribution of possible worlds; and 2) it treats single-alternative and multi-alternative x-tuples differently because probabilities are ignored completely for single-alternative x-tuples. In the next chapter, we propose a new approach for matching x-tuples in the x-relation data model. Our approach avoids the above mentioned problems. In Chapter 4, we extend our model and algorithms for distributed systems.

The only proposal [102] for merging x-tuples in the x-relation data model does not merge tuples with null values, which results in not aggregating the whole data about an entity into one tuple. Moreover, the amount of data quality improvement, which is achieved by the only proposal [94] for dealing with the deduplication problem in the x-relation data model, may not be noticeable, which is in contrast to the aim of deduplication. In Chapter 5, we propose a new approach for the deduplication problem in the x-relation data model, which does not suffer from the above shortcomings.

Finally, in Chapter 6, we propose an approach for automatic schema matching, which arises in many applications that need to deal with the entity resolution problem.