Entity Resolution for Probabilistic Data

Ayat Naser, Reza Akbarinia, Hamideh Afsarmanesh, Patrick Valduriez

Entity Resolution for Probabilistic Data

Naser Ayat[#1], Reza Akbarinia[*3], Hamideh Afsarmanesh[#2], Patrick Valduriez[*4]

[#]Informatics Institute, University of Amsterdam, Amsterdam, Netherlands
[1]s.n.ayat@uva.nl, [2]h.afsarmanesh@uva.nl
[*]INRIA and LIRMM, Montpellier, France
[3,4]{Firstname.Lastname@inria.fr}

Abstract. Entity resolution is the problem of identifying the tuples that represent the same real world entity. In this paper, we address the problem of entity resolution over probabilistic data (ERPD), which arises in many applications that have to deal with probabilistic data. To deal with the ERPD problem, we distinguish between two classes of similarity functions, i.e. context-free and context-sensitive. We propose a PTIME algorithm for context-free similarity functions, and a Monte Carlo approximation algorithm for context-sensitive similarity functions. We also propose improvements over our proposed algorithms. We validated our algorithms through experiments over both synthetic and real datasets. Our extensive performance evaluation shows the effectiveness of our algorithms.

Keywords: Entity resolution, Probabilistic database

1 INTRODUCTION

In recent years, we have been witnessing much interest in uncertain data management in many application areas such as data integration [1, 2], sensor networks [3], information extraction [4], etc. Much research effort has been devoted to several aspects of uncertain data management, including data modeling [5, 6], skyline queries [7], top-k queries [8–11], nearest neighbor search [12], XML documents [13], etc. Untrusted sources, imprecise measuring instruments, and uncertain methods, are some reasons that cause uncertainty in data. The main difference between a traditional *certain* database and an uncertain database is that an uncertain database represents a set of possible database instances, rather than a single one. Recent uncertain data models tend to represent data uncertainty with probabilistic events [14, 5, 6].

An important problem that arises in many applications such as information integration is that of Entity Resolution (ER) [15]. ER is the process of identifying tuples that represent the same real-world entity. For instance, consider the two tuples ("Thomas Michaelis", "45, Main street") and ("T. Michaelis", "45, Main st.") on the schema S(name, address) that represent the same person with different conventions. An ER solution is expected to detect that both tuples represent the same entity. The problem of ER is challenging since the same entity can be encoded in different ways due to a variety of reasons such as different formatting conventions, abbreviations, and typographic errors. It has been well studied in the literature for certain data (refer to [15] for a survey), but it has not been deeply investigated for probabilistic data. Before discussing existing solutions, let us first motivate the need for ER in probabilistic data (which we call ERPD), with two examples from scientific data management and anti-criminal domains.

Example 1: Finding astronomical objects in astrophysics data. In astrophysics, as well as in other scientific disciplines, the correlation and integration of observational data is the key for gaining new scientific insights. Astronomical observatories produce data about sky surveys, most of which is probabilistic [16]. In its simplified form, an observatory maintains a single probabilistic relation *Objects* that contains data about the observed *astronomical objects* in the sky surveys. Each *object* is represented using a number of alternative tuples each with a membership probability showing its degree of certainty which is computed using some rules. The alternatives are disjoint, meaning that at most one of them can be true. The uncertainty model, which is used in this example, has been widely used in the database literature for representing probabilistic data, e.g. [5, 10, 17, 18], and also for representing astrophysics data [19]. The astrophysics researchers who want to track a particular astronomical object, which has been represented by an uncertain entity *e*, are very interested in queries like the following: *among astronomical objects observed in a sky region, find the object which is most probably the same object as e.*

Example 2: Suspect detection in anti-criminal police database. The anti-criminal police is faced with many crimes every year. It spends a lot of time and money gathering data about every crime from different sources such as witnesses, interrogations, and police's informants. Some of the gathered data are not certain, for some reasons, e.g. the police cannot completely trust informants and witnesses. To represent this uncertainty, probability values can be attached to the data to show their likelihood of truth according to the confidence on the sources. These probabilistic data are used to find possible suspects, and can greatly speed up the investigation process. In a very simplified form, the police maintains a single relation *Suspects* that contains data about suspects. In this relation, each individual suspect is represented using an entity consisting of a number of alternative tuples each associated with a probability value showing its likelihood of truth. When a crime occurs, detectives gather data about the perpetrator, and the gathered data can be represented in the form of an uncertain entity, say e. To get more information about the perpetrator represented by e, detectives are interested in the following query: *find in the uncertain database the person who is most probably the same person as e.* Notice that if there is more than one perpetrator, the police can represent each one using an uncertain entity and repeat the process for each entity.

Existing proposals for the ER problem are not applicable to the above examples since they ignore probability values completely and return the most similar tuples as the solution. To the best of our knowledge, the works presented in [20] and [21] are the only proposals for dealing with the ERPD problem. The proposal in [20] ignores the probability values and the proposal in [21] uses the probability values only for normalizing the similarity of the uncertain entities. These proposals do not define the ERPD problem based on both similarity and probability of tuples. Consequently, they are not appropriate for answering the queries issued in the above examples.

Inspired by the literature on uncertain data management, in this paper we adopt the well-known possible worlds semantics for defining the semantics for the ERPD problem and propose efficient algorithms for computing it. However, developing an efficient solution for the ERPD problem is challenging, particularly due to the following reasons. First, we must take into account two parameters for matching the entities: the similarity and the probability values. Second, due to the uncertainty in the entity e, there may be different similarity values between e and the tuples of D. Third, in the case of *context-sensitive* similarity functions (see the definition in Section 3), the similarity of two entities may be different in different possible worlds, i.e. in different instances of the database. A naïve solution for ERPD involves enumerating all possible worlds of the uncertain entity and the database. However, this solution is exponential in the number of tuples of D.

In this paper, we address the ERPD problem and propose a complete solution for it. Our contributions are summarized as follows:

- We adapt the possible worlds semantics of probabilistic data to define the problem of ERPD based on both similarity and probability of tuples.
- We propose a PTIME algorithm for the ERPD problem. This algorithm is applicable to a large class of the similarity functions, i.e. *context-free* functions. For the rest of similarity functions (i.e. *context-sensitive*), we propose a Monte Carlo approximation algorithm.
- We deal with the problem of significant setup time in existing context-sensitive functions, which makes them very inefficient for the Monte Carlo algorithm. We propose a new efficient context-sensitive similarity function that is very appropriate for the Monte Carlo algorithm.
- We propose a parallel version of our Monte Carlo algorithm using the MapReduce framework.

We conducted an extensive experimental study to evaluate our approach for ERPD over both real and synthetic datasets. The results show the effectiveness of our algorithms. To the best of our knowledge, this is the first study of the ERPD problem that adopts the possible world semantics and develops efficient algorithms for it.

The rest of the paper is organized as follows. In Section 2, we present our probabilistic data model, and define the problem we address. In Section 3, we propose our solution for the ERPD problem for context-free similarity functions. In section 4, we propose our solution for dealing with the ERPD problem when the similarity function is context-sensitive. In section 5, we report the performance evaluation of our techniques over synthesis and real data sets. Section 6 discusses related work. Section 7 concludes.

entity	tuple	Age	Height	Weight	Eye color	P
e	$t_{e,1}$	35-40	175-180	90+	Gray	0.4
	$t_{e,2}$	25-30	185-190	-	Blue	0.5

(a)

entity	tuple	Name	Age	Height	Weight	Gender	Eye color	P
e_1	$t_{1,1}$	N1	38	178	70	M	Gray	0.6
	$t_{1,2}$	N1	36	168	80	M	Hazel	0.4
e_2	$t_{2,1}$	N2	36	180	75	F	Blue	0.4

(b)

World w	P(w)
$\{t_{1,1}\}$	$0.6 \times (1 - 0.4) = 0.36$
$\{t_{1,2}\}$	$0.4 \times (1 - 0.4) = 0.24$
$\{t_{1,1}, t_{2,1}\}$	$0.6 \times 0.4 = 0.24$
$\{t_{1,2}, t_{2,1}\}$	$0.4 \times 0.4 = 0.16$

(c)

Fig. 1. a) Example of an entity in the x-tuple model, b) Example of x-relation *Suspects*, c) the possible worlds of *Suspects*

2 PROBLEM DEFINITION

In this section, we first give our assumptions regarding the probabilistic data model. Then, we define the problem which we address.

2.1 probabilistic Data Model

For representing an uncertain entity, we use the x-tuple data model [22], and represent each entity as an x-tuple. In each x-tuple there are several possible tuples, called alternatives. Each alternative is associated with a probability value showing its likelihood of truth. The alternatives are mutually exclusive, meaning that at most one of them can be true, and the sum of the confidence values of the alternatives is less than or equal to one. As an example of x-tuple, consider the entity e in Figure 1(a). Notice that either $t_{e,1}$ or $t_{e,2}$ is true.

The uncertain database $\mathcal{D}$ (called also x-relation) consists of a set of pairwise disjoint x-tuples that are subject of independent probabilistic events. Formally,

$$\mathcal{D} = \{e_1, \ldots, e_N\}, e_i = \{t_{i,1}, \ldots, t_{i,|e_i|}\}, e_i \neq \emptyset, e_i \cap e_j = \emptyset, i, j \in [1..N].$$

We use D to denote the set of all tuples in $\mathcal{D}$. Figure 1(b) shows an example of x-relation, named *Suspects*. In the *Suspects* relation, $t_{1,1}$ and $t_{1,2}$ together form one x-tuple, and $t_{2,1}$, by itself, form another x-tuple.

An uncertain database $\mathcal{D}$ can be interpreted as the set of all possible certain database instances, called possible worlds and denoted by $PW(\mathcal{D})$, each with a probability of occurrence. Figure 1(c) shows the possible worlds set of the uncertain database *Suspects* and the probability of each member of this set.

We define the possible worlds set of an uncertain entity e and an uncertain database $\mathcal{D}$, denoted by $PW(e, \mathcal{D})$, as follows:

$$PW(e, \mathcal{D}) = \{w \mid w = \{t\} \times v, t \in e, v \in PW(\mathcal{D})\}.$$

Notice that in each possible world $w \in PW(e, \mathcal{D})$ only one alternative of the uncertain entity e is valid. As an example, Figure 2(b) shows all eight members of *PW(e, Suspects)* together with their probability of occurrence. Notice that the members of each possible world $w \in PW(e, \mathcal{D})$ are tuple pairs in the from of (t, t'), where $t \in e$ and $t' \in D$.

2.2 Problem Statement

In a *certain* database, the problem of entity resolution is defined as follows [23]: given an entity (e.g. tuple) e and a database D, find $t_{max} \in D$ such that t_{max} has the maximum similarity to e. While the problem of finding t_{max} is semantically clear in certain data, its semantics is not clear in probabilistic data since we have to consider two parameters: similarity and probability. In this section, we use the possible worlds semantics for defining the semantics of the ERPD problem.

Let us first consider the case where the entity e has a single alternative. In this case, the probability that a tuple $t \in D$ be the most similar to e, say $P_{MSP}(t)$, is equal to the cumulative probability of the possible worlds where t is t_{max}, i.e. the most similar tuple to e. We denote the tuple with maximum P_{MSP} as the *most-probable match tuple* for entity e.

In the case where the entity e has multiple alternatives, the most-probable match tuple needs to be extended by the alternative of e with which it is the most-probable match tuple, say pair $(t, t'), t \in e, t' \in D$. We refer to this pair as *most-probable match pair* and we use the first element of the pair for tuples in entity e and the second element for tuples in D. In addition to the most-probable match pair concept, uncertain data enables us to define another new

Pair	Rank
$(t_{e,1}, t_{2,1})$	1
$(t_{e,1}, t_{1,1})$	2
$(t_{e,2}, t_{2,1})$	3
$(t_{e,2}, t_{1,2})$	4
$(t_{e,1}, t_{1,2})$	5
$(t_{e,2}, t_{1,1})$	6

(a)

w	PW(e, Suspects)	MSP(w)	P(w)
w_1	$\{(t_{e,1}, t_{1,1})\}$	$(t_{e,1}, t_{1,1})$	$0.4 \times 0.36 = 0.144$
w_2	$\{(t_{e,1}, t_{1,2})\}$	$(t_{e,1}, t_{1,2})$	$0.4 \times 0.24 = 0.096$
w_3	$\{(t_{e,1}, t_{1,1}), (t_{e,1}, t_{2,1})\}$	$(t_{e,1}, t_{2,1})$	$0.4 \times 0.24 = 0.096$
w_4	$\{(t_{e,1}, t_{1,2}), (t_{e,1}, t_{2,1})\}$	$(t_{e,1}, t_{2,1})$	$0.4 \times 0.16 = 0.064$
w_5	$\{(t_{e,2}, t_{1,1})\}$	$(t_{e,2}, t_{1,1})$	$0.5 \times 0.36 = 0.18$
w_6	$\{(t_{e,2}, t_{1,2})\}$	$(t_{e,2}, t_{1,2})$	$0.5 \times 0.24 = 0.12$
w_7	$\{(t_{e,2}, t_{1,1}), (t_{e,2}, t_{2,1})\}$	$(t_{e,2}, t_{2,1})$	$0.5 \times 0.24 = 0.12$
w_8	$\{(t_{e,2}, t_{1,2}), (t_{e,2}, t_{2,1})\}$	$(t_{e,2}, t_{2,1})$	$0.5 \times 0.16 = 0.08$

(b)

Pair	P_{MSP}
$(t_{e,2}, t_{2,1})$	$P(w_7)+P(w_8) = 0.2$
$(t_{e,2}, t_{1,1})$	$P(w_5) = 0.18$
$(t_{e,1}, t_{2,1})$	$P(w_3)+P(w_4) = 0.16$
$(t_{e,1}, t_{1,1})$	$P(w_1) = 0.144$
$(t_{e,2}, t_{1,2})$	$P(w_6) = 0.12$
$(t_{e,1}, t_{1,2})$	$P(w_2) = 0.096$

(c)

Entity	P_{MSP}
e_1	$P(w_1)+P(w_2)+P(w_5)+P(w_6) = 0.54$
e_2	$P(w_3)+P(w_4)+P(w_7)+P(w_8) = 0.36$

(d)

Fig. 2. a) Tuple pairs ranked based on their similarity; b) possible worlds space of e and $Suspects$, MSP in each world; c) all pairs and their probability to be MSP (P_{MSP}); d) entities and their P_{MSP}

concept, which does not make sense in *certain* data: *most-probable match entity*. The most-probable match entity e is an entity in $\mathcal{D}$ that has the highest probability of being most similar to e. Also to avoid repetition, we refer to the two concepts of the most-probable match pair and the most-probable match entity as *most-probable matches*. Let us intuitively explain these concepts using an example.

Example 3. Consider the entity e and the *Suspects* database in the Figures 1(a) and 1(b) respectively. Let F be a similarity function that ranks all possible tuple pairs of e and $Suspects$ as they appear in Figure 2(a). Figure 2(b) shows the eight possible worlds of $PW(e, Suspects)$, the probability of each world, and the most similar pair (MSP) in each world. The result of computing P_{MSP} for all pairs is shown in Figure 2(c). For instance, pair $(t_{e,1}, t_{2,1})$ is MSP in w_3 and w_4, thus, the cumulative probability that $(t_{e,1}, t_{2,1})$ is the most similar pair (denoted as P_{MSP}), is the sum of the probabilities of w_3 and w_4. We observe that pair $(t_{e,2}, t_{2,1})$ is the *most-probable match pair* since it has the maximum P_{MSP} among all other pairs. Figure 2(c) shows the result of computing P_{MSP} for all entities in the *Suspects* database. This figure shows that e_1 is the *most-probable match entity* for entity e since it has the maximum P_{MSP} among all entities in the database.

We now formally define the concept of most-probable matches.

Definition 1. ***Most-probable matches**:*
Let $\mathcal{D}$ be an uncertain database on schema S_D. Let e be an uncertain entity on schema S_e and $S_e \subseteq S_D$. Let $w \in PW(e, \mathcal{D})$ be a possible world and $P(w)$ be the probability that w occurs. Let F be a similarity function for computing the similarity between tuples. Let the most similar pair of a world w, denoted as $MSP(w)$, be the pair that has the maximum similarity value among the pairs in w according to similarity function F. Let ρ be an tuple pair in $e \times D$. Let $P_{MSP}(\rho)$ be the aggregated probability that pair ρ is the most similar pair in $PW(e, \mathcal{D})$, i.e.

$$P_{msp}(\rho) = \sum_{w \in PW(e,\mathcal{D}) \wedge \rho = MSP(w)} P(w)$$

*Then, we define the **most probable match pair** of e and $\mathcal{D}$, $MPMP(e, \mathcal{D})$, as:*

$$MPMP(e, \mathcal{D}) = \arg\max_{\rho \in e \times D} P_{MSP}(\rho)$$

*and the **most-probable match entity** of e and $\mathcal{D}$, $MPME(e, \mathcal{D})$, as:*

$$MPME(e, \mathcal{D}) = \arg\max_{e_i \in \mathcal{D}} \sum_{\rho = (t,t') \in e \times D, t' \in e_i} P_{MSP}(\rho)$$

As mentioned earlier, we refer to these concepts as *most-probable matches*.

The entity resolution approach strongly depends on the given similarity function. The similarity functions in the literature can be categorized into two classes: *context-free* and *context-sensitive*. In a context-free similarity function, the similarity value of two tuples only depends on their attribute values. Jaro-Winkler [24], Monge-Elkan [25, 26] and

Levenshtein [27] are examples of context-free similarity functions. On the other hand, in a context-sensitive similarity function, the similarity value of two tuples depends on the attribute values of the two tuples and also on the relation to which they belong. This means that the similarity of two tuples may change when the other tuples of the relation change. TFIDF [28] and Ranked-List-Merging [29] are examples of context-sensitive similarity functions.

Given uncertain entity e, uncertain database $\mathcal{D}$, and similarity function F, our goal is to efficiently find the most-probable matches. In Sections 3 and 4, we present our approach for finding the most-probable matches, respectively, for the context-free and context-sensitive similarity functions.

3 Context-free Entity Resolution

In this section, we consider a context free similarity function for the ERPD problem, and propose an efficient algorithm for dealing with this problem. Then, we present two improved versions of our algorithm.

The straightforward approach for computing the most-probable matches is to enumerate $PW(e, \mathcal{D})$. This approach does not scale well due to the exponential number of members of set $PW(e, \mathcal{D})$.

In the case of context-free similarity function, the similarity score of a tuple pair remains constant in all possible worlds where the tuple pair appears. We rely on this fact to efficiently compute the most-probable matches without enumerating the possible worlds set $PW(e, \mathcal{D})$.

Let S be the set of all tuple pairs, i.e. $S = e \times D$. Let $\rho = (t, t_i)$ be a tuple pair in set S. Since alternative tuples of e are mutually exclusive, there is no possible world in $PW(e, \mathcal{D})$ containing pair ρ together with pair $\rho' = (t', t_j)$, where $\rho' \in S$, and $t \neq t'$. Thus, to compute $P_{MSP}(\rho)$, we just have to consider the subset of tuple pairs in S which have t as their first elements, i.e. set $\{t\} \times D$. We use this fact to compute $P_{MSP}(\rho)$.

Let $L = \{t_1, \ldots, t_n\}$ be the list of D tuples sorted based on their similarity with $t \in e$ in descending order. Let (t, t_i) be a tuple pair where t_i is the tuple which lies in the ith index of the sorted list L. We can calculate $P_{MSP}(t, t_i)$ as the intersection of two independent events: t occurs; and among tuples t_1 to t_i, only t_i occurs. Considering x-tuple correlations in calculating the probability of the latter event, we can calculate $P_{MSP}(t, t_i)$ as

$$P_{MSP}(t, t_i) = P(t) \times P(t_i) \times \prod_{x \in X_i} (1 - P(x)) \tag{1}$$

where P is the occurrence probability of a tuple or x-tuple, and X_i is the set of x-tuples formed by considering x-tuple correlations between the tuples t_1 to t_{i-1} while the x-tuple containing t_i is omitted from it. Using (1), the context-free algorithm, denoted as CFA algorithm, computes P_{MSP} of all tuple pairs in set S and then use the computed P_{MSP}s to compute the most-probable matches.

Algorithm 1 describes the details of our method for computing the most-probable matches. This algorithm takes as input an uncertain x-relation database $\mathcal{D}$, uncertain x-tuple entity e, and context-free similarity function *Sim*. Steps 4-11 are repeated for every alternative t of e. Step 4 computes the similarity score of t to every tuple of D; sorts the tuples based on their similarity scores; and stores the result in list L. For each tuple $L[i]$ in list L, steps 6-9 compute $P_{MSP}(t, L[i])$. Using (1), we know that $P_{MSP}(t, L[i])$ is the intersection of two independent events: t occurs; and among tuples $L[1]$ to $L[i]$ only $L[i]$ occurs. For computing the probability of the latter event, the correlated tuples have to be grouped together. This is done by steps 6-8. Step 6 puts the tuples $L[1]$ to $L[i]$ in the set T and step 7 obtains the intersection of every x-tuple in $\mathcal{D}$ with the members of T. The resulted set, i.e. X, contains the tuples of T grouped into a number of x-tuples. Step 8 finds the x-tuple in X that includes tuple $L[i]$ and removes it from X. Using equation (1), Step 9 computes the P_{MSP} of the pair $(t, L[i])$. When the algorithm finishes computing the P_{MSP} of all pairs, step 13 finds the most-probable matches. This step finds the most-probable match pair by finding the record in set R with the maximum value for P_{MSP} field. The most-probable match entity can be computed by aggregating the P_{MSP} field for every entity $e_i \in \mathcal{D}$ and then finding the entity with maximum aggregated P_{MSP} value.

Let us analyze the execution time of CFA algorithm. Let n be the number of tuples involved in D, and m be the number of alternatives of e. Step 4 takes $O(n)$ time for computing the similarity scores and $O(n \times \log n)$ for sorting the scores. An efficient implementation of steps 6-8 takes $O(i + N)$ time, where N is the number of x-tuples in $\mathcal{D}$. Thus, steps 5-8 take $\sum_{i=1}^{n} O(i + N)$ time, which is $O(n^2)$ since $N < n$. Therefore, steps 4-8 are done in $O(n^2)$. Since these steps are repeated m times (i.e. the number of alternatives of e), steps 3-8 take $O(m \times n^2)$ time. Step 9 takes $O(m \times n)$ time, which is dominated by $O(m \times n^2)$. Thus, the execution cost of the algorithm is $O(m \times n^2)$.

Algorithm 1 Computing most-probable matches for the case of context-free similarity function

Input:
- Uncertain database $\mathcal{D}$
- Uncertain entity e
- Context-free similarity function *Sim*

Output: Most-probable matches

define list L
define set R and $R \leftarrow \emptyset$
for all $t \in e$ **do**
  sort D tuples based on $Sim(t, t_i)$, $t_i \in D$, in descending order and store the result in list L
  for all $i \in [1..|L|]$ **do**
    $T \leftarrow \{L[1], \ldots, L[i]\}$
    $X \leftarrow \{x \mid x = x' \cap T, x' \in \mathcal{D}, x \neq \emptyset\}$
    $X \leftarrow X - \{x \mid x \in X, x \cap L[i] \neq \emptyset\}$
    $P_{MSP} \leftarrow P(L[i]) \times P(t) \times \prod_{x \in X} \left(1 - \sum_{t \in x} P(t)\right)$
    add record $(t, L[i], P_{MSP})$ to set R
  end for
end for
find the most-probable matches using set R and return them

3.1 Improving CFA Algorithm

In this section, we improve the efficiency of the CFA algorithm by computing the P_{MSP} of a subset of pairs in set $e \times D$. We consider two versions of the CFA algorithm as follws:

- CFA-MPMP: this version only computes MPMP$(e, \mathcal{D})$, i.e. the most-probable match pair.
- CFA-MPME: this version only computes MPME$(e, \mathcal{D})$, i.e. the most-probable match entity.

CFA-MPMP Algorithm Consider the sorted list L in the CFA algorithm. The core idea in improving CFA is that after computing $P_{MSP}(t, L[i])$, we compute the upper bound on the P_{MSP} values that we obtain if we continue processing list L. If the computed upper bound is smaller than the maximum P_{MSP} value which has been computed so far, then we stop processing list L. We denote this upper bound as $Potential(t, L[i])$. The following lemma computes the Potential of a tuple pair.

Lemma 1. *Let $L[i]$ be the tuple which lies in the ith location of list L. Let Y_i be the set of x-tuples formed by considering correlations between the tuples $\{L[1], \ldots, L[i]\}$, then*

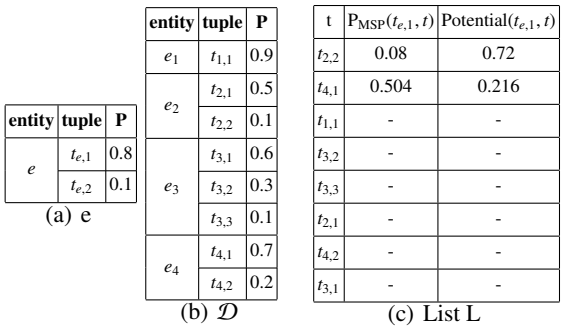
$$Potential\,(t, L[i]) = P(t) \times \prod_{x \in Y_i} (1 - P(x)).$$

Proof. Based on its definition, $Potential(t, L[i])$ is an upper bound on $P_{MSP}(t, t')$, where t' is a tuple which comes after $L[i]$ in list L. We show that $P_{MSP}(t, t')$ is maximized when $t' = L[i+1]$ and $P(t') = 1$.

Let W be the set $\{w \mid w \in PW(e, \mathcal{D}), MSP(w) = (t, L[j]), j > i\}$ and $P(W)$ be the probability that a member of W occurs. Since $P(t') = 1$, it is clear that tuple pair (t, t') exists in all members of W and is MSP in each of them. Thus, $P_{msp}(t, t')$ is equal to $P(W)$ which is the maximum possible value for $P_{msp}(t, L[j])$, where $j > i$. Using (1) and setting the value of $P(L[i+1])$ to one, we have

$$P_{MSP}(t, L[i+1]) = P(t) \times \prod_{x \in Y_i} (1 - P(x)) = RHS\,(1)$$

where Y_i is the set of x-tuples formed by considering correlations between the tuples $\{L[1], \ldots, L[i]\}$.



entity	tuple	P
e	$t_{e,1}$	0.8
	$t_{e,2}$	0.1

(a) e

entity	tuple	P
e_1	$t_{1,1}$	0.9
e_2	$t_{2,1}$	0.5
	$t_{2,2}$	0.1
e_3	$t_{3,1}$	0.6
	$t_{3,2}$	0.3
	$t_{3,3}$	0.1
e_4	$t_{4,1}$	0.7
	$t_{4,2}$	0.2

(b) $\mathcal{D}$

t	$P_{MSP}(t_{e,1}, t)$	Potential$(t_{e,1}, t)$
$t_{2,2}$	0.08	0.72
$t_{4,1}$	0.504	0.216
$t_{1,1}$	-	-
$t_{3,2}$	-	-
$t_{3,3}$	-	-
$t_{2,1}$	-	-
$t_{4,2}$	-	-
$t_{3,1}$	-	-

(c) List L

Fig. 3. An example of uncertain entity e, database D, and the improvement method

Let us illustrate the CFA-MPMP algorithm using an example. Consider uncertain entity e and uncertain database $\mathcal{D}$ shown in Figures 3(a) and 3(b) respectively. Let *Sim* be a context-free similarity function. Let L, shown in Figure 3(c), be the list of D's tuples sorted based on their similarity with $t_{e,1}$, where the similarity scores are computed using *Sim*.

Visiting the first tuple in L, the $P_{MSP}(t_{e,1}, t_{2,2})$ is equal to the probability of the event that $t_{e,1}$ and $t_{2,2}$ occur, thus, $P_{MSP}(t_{e,1}, t_{2,2})$ is equal to $P(t_{e,1}) \times P(t_{2,2}) = 0.08$. The *Potential* of this pair is equal to the probability of the event that $t_{e,1}$ occurs but $t_{2,2}$ does not occur, which is equal to $P(t_{e,1}) \times (1 - P(t_{2,2})) = 0.72$. This is an upper bound on the P_{MSP} values that we obtain if we continue processing list L. Since the computed *Potential* is greater than the so far computed maximum P_{MSP} value, i.e. 0.08, we continue processing the list.

Visiting the second tuple in L, the $P_{MSP}(t_{e,1}, t_{4,1})$ is equal to the probability of the event that $t_{e,1}$ and $t_{4,1}$ occur but $t_{2,2}$ does not occur, which is equal to $P(t_{e,1}) \times P(t_{4,1}) \times (1 - P(t_{2,2})) = 0.504$. The *Potential* of this pair is equal to the probability of the event that $t_{e,1}$ occurs but neither $t_{2,2}$ nor $t_{4,1}$ occurs, which is equal to $P(t_{e,1}) \times (1 - P(t_{2,2})) \times (1 - P(t_{4,1})) = 0.216$. At this point, we stop processing the list since the computed *Potential* is less than the so far computed maximum P_{MSP} value, i.e. 0.504.

It is not necessary to consider the tuple pairs whose first element is $t_{e,2}$ because the upper bound on their P_{MSP} values is $P(t_{e,2}) = 0.1$, which is less than the so far computed maximum P_{MSP}. Thus, $P_{MSP}(t_{e,1}, t_{4,1})$ is maximum among all pairs, and $(t_{e,1}, t_{4,1})$ is $MPMP(e, \mathcal{D})$. Notice that we only process two pairs out of the whole 16 pairs to compute $MPMP(e, \mathcal{D})$.

CFA-MPME Algorithm We know that sum of the P_{MSP} of all pairs is equal to sum of the probabilities of e's alternatives, i.e. $\sum_{\rho \in e \times D} P_{MSP}(\rho) = \sum_{t \in e} P(t)$. We use this fact to stop the CFA algorithm when the so far computed most-probable match entity (i.e. MPME) remains the same even if we continue the algorithm.

Let us define the stop condition more precisely. Let C be the set of pairs that we have computed their P_{MSP}s so far. Let e_i be an entity in $\mathcal{D}$ and $P_{agg}(e_i)$ be its so-far computed aggregated probability to be MPME, i.e.

$$P_{agg}(e_i) = \sum_{(t,t') \in C, t' \in e_i} P_{MSP}(t, t')$$

let e_{max} be the entity with maximum P_{agg} value. Then, the stop condition can be written as follows:

$$\forall e_i \in \mathcal{D}, P_{agg}(e_i) + \left(\sum_{t \in e} P(t) - \sum_{\rho \in C} P_{MSP}(\rho) \right) < P_{agg}(e_{max})$$

We check the stop condition after computing P_{MSP} in Step 9 of Algorithm 1, and finish the algorithm if the condition is met.

As an example consider uncertain entity e and uncertain database $\mathcal{D}$ shown in Figures 3(a) and 3(b) respectively. The sum of the probabilities of e's alternatives is equal to 0.9. Visiting the first two pairs of list L, shown in Figure 3(c), and computing $P_{MSP}(t_{e,1}, t_{2,2})$ and $P_{MSP}(t_{e,1}, t_{4,1})$, we have: $P_{agg}(e_1) = 0$, $P_{agg}(e_2) = 0.08$, $P_{agg}(e_3) = 0$, and

$P_{agg}(e_4) = 0.504$. The sum of so far computed P_{MSP}s is equal to 0.584 and the sum of uncomputed P_{MSP}s is equal to $0.9 - 0.584 = 0.316$. The entity e_4 is MPME since even if all uncomputed P_{MSP}s are from an entity other than e_4, its P_{agg} is still less than that of e_4.

4 Context-sensitive Entity Resolution

In this section, we consider a context-sensitive similarity function for the ERPD problem, and present a Monte Carlo approach for approximating the most-probable matches. Then, we propose two solutions for improving the performance of the Monte Carlo algorithm.

4.1 Monte Carlo Algorithm

When the similarity function is context-sensitive, the similarity between two tuples may change when the contents of the database changes. For instance when using TFIDF, adding or removing tuples to/from database, may change the frequency of the *terms* in the database, which in turn changes the similarity score of two tuples. This means that the similarity of two tuples in different possible worlds does not remain constant. Therefore, we cannot use the CFA algorithm for computing most-probable matches.

In the absence of an efficient exact algorithm for ERPD in the case of context-sensitive similarity functions, we use the randomized algorithm *Monte Carlo (MC)* for approximating the answer. The MC algorithm repeatedly chooses at random a possible world and an alternative of the uncertain entity, and computes the tuple pair that is the most similar. For each pair $\rho = (t, t')$, the probability $P_{MSP}(\rho)$ of being the most similar match, is approximated by $P'_{MSP}(\rho)$, which is the fraction of times that t was the most similar to t' in the sampled possible worlds. The MC algorithm guarantees that after sampling M possible worlds, $P'_{MSP}(\rho)$ is in the interval $\left[P_{MSP}(\rho) - \frac{z\delta}{\sqrt{M}}, P_{MSP}(\rho) + \frac{z\delta}{\sqrt{M}}\right]$ with the confidence $1 - \delta$, where $P\{-z \leq N(0,1) \leq z\} = 1 - \delta$, $N(0,1)$ is the standard normal distribution, and δ is the deviation of the distribution.

Notice that the probabilities of the individual possible worlds are taken into account in the way that a high probable possible world is chosen with a higher likelihood than a less probable world.

4.2 Parallel MC

In this section, we improve the performance of the MC algorithm by proposing a parallel version of this algorithm, denoted as MC-MapReduce, using the MapReduce framework.

MapReduce provides a framework for performing a two-phase distributed computation on large datasets. In the *Map* phase, the system partitions the input dataset into a set of disjoint units (denoted as input splits) which are assigned to workers, known as mappers. In parallel, each mapper scans its input split and applies a user-specified map function to each record in the input split. The output of the user's map function is a set of $\langle key, value \rangle$ pairs which are collected for MapReduce's *Reduce* phase. In the reduce phase, the key-value pairs are grouped by key and are distributed to a series of workers, called reducers. Each reducer then applies a user-specified reduce function to all the values for a key and outputs a final value for the key. The collection of final values from all of the reducers is the final output of MapReduce.

Given an uncertain entity e, an uncertain database $\mathcal{D}$, and M as the number of iterations in the MC algorithm, the idea of our MC-MapReduce algorithm is to ask M mappers to do one iteration of the MC algorithm in parallel, then collect the results of mappers in the reduce phase, and compute the most-probable matches.

MC-MapReduce algorithm works as follows. It gets the uncertain entity e, uncertain database $\mathcal{D}$, and the required parameters of the MC algorithm (e.g. δ and ϵ) as input and computes M as the number of possible worlds that it should sample to obtain the desired confidence. Then, MC-MapReduce assigns M mappers to do the map function.

Algorithm 2 shows the pseudo code of the map function. This algorithm gets e and $\mathcal{D}$, and a context-sensitive similarity function Sim as input. Steps 1-2 generate an alternative of e, say t, and a possible world of $\mathcal{D}$, say W, at random. Step 3 computes the most-similar tuple of W to t, say t_{max}. Steps 4-6 return the key-value pair $\langle (t, t_{max}), 1 \rangle$ as the output of the map function; meaning that the pair (t, t_{max}) has been the most-similar pair in one possible world.

Algorithm 2 Map function

Input:

- $\langle e, \mathcal{D} \rangle$, where e is an uncertain entity, and $\mathcal{D}$ is an uncertain database
- context-sensitive similarity function *Sim*

1: generate an alternative t of e at random
2: generate a possible world W of $\mathcal{D}$ at random
3: $t_{max} \leftarrow \arg\max_{t' \in W} Sim(t, t')$
4: $key \leftarrow (t, t_{max})$
5: $value \leftarrow 1$
6: Emit $\langle key, value \rangle$

Algorithm 3 Reduce function

Input:

- key (t, t'), where $t \in e, t' \in D$
- value set V

1: $key \leftarrow (t, t')$
2: $value \leftarrow |V|$
3: Emit $\langle key, value \rangle$

Algorithm 4 Finalize

Input: set S of key-value pairs $\langle (t, t'), v \rangle$
Output: most-probable matches

1: $M \leftarrow \sum_{\langle (t,t'),v \rangle \in S} v$
2: **for all** pair $\langle (t, t'), v \rangle \in S$ **do**
3: $\quad P_{(t,t')} \leftarrow v/M$
4: **end for**
5: compute most-probable matches using P

The MapReduce framework receives all generated pairs and sends all pairs with the same key, i.e. (t, t'), and the set of their values, i.e. 1, to one reducer function. Algorithm 3 shows the pseudo code of the reduce function. This algorithm gets (t, t') as key and the set of values V, which all of its members are 1, as input. Then, it simply counts the number of members of V and generates the final key-value pair $\langle (t, t'), |V| \rangle$ as the output of the reduce function.

Algorithm 4 shows the steps which are performed by MC-MapReduce after all mappers and reducers finish their task. This algorithm gets all key-value pairs $\langle (t, t'), v \rangle$ as input. Then, it approximates P_{MSP} of tuple pair (t, t') by dividing the number of times that it has been the most-similar pair, i.e. v, by M, i.e. the sum of occurrence of all tuple pairs. Using these probabilities, MC-MapReduce approximately computes the most-probable matches.

4.3 CB Similarity Function

In this section, we improve the performance of the MC algorithm by proposing a new context-sensitive similarity function appropriate for MC computation. As we will show in Section 5, our similarity function significantly outperforms the baseline context-sensitive similarity functions in terms of success rate and response time.

To the best of our knowledge, all context-sensitive similarity functions in the literature need a *setup* time for at least one pass over the database to compute some statistical features of the data. For example TFIDF similarity function

	Gender	City	Age-range
t	F	A	18-40

(a)

	Gender	City	Age-range
t_1	M	B	18-40
t_2	M	B	0-17
t_3	F	B	18-40
t_4	F	B	41-64
t_5	F	B	41-64
t_6	F	A	18-40
t_7	F	A	18-40
t_8	F	A	41-64
t_9	M	A	65+
t_{10}	M	C	18-40

(b)

	Smallest community	Similarity
t_1	$\{t_1,t_3,t_6,t_7,t_{10}\}$	$1 - (\log 5/\log 10) = 0.3$
t_2	$\emptyset$	0
t_3	$\{t_3,t_6,t_7\}$	$1 - (\log 3/\log 10) = 0.52$
t_4	$\{t_3,t_4,t_5,t_6,t_7,t_8\}$	$1 - (\log 6/\log 10) = 0.22$
t_5	$\{t_3,t_4,t_5,t_6,t_7,t_8\}$	$1 - (\log 6/\log 10) = 0.22$
t_6	$\{t_6,t_7\}$	$1 - (\log 2/\log 10) = 0.7$
t_7	$\{t_6,t_7\}$	$1 - (\log 2/\log 10) = 0.7$
t_8	$\{t_6,t_7,t_8\}$	$1 - (\log 3/\log 10) = 0.52$
t_9	$\{t_6,t_7,t_8,t_9\}$	$1 - (\log 4/\log 10) = 0.4$
t_{10}	$\{t_1,t_3,t_6,t_7,t_{10}\}$	$1 - (\log 5/\log 10) = 0.3$

(c)

Fig. 4. a) An example tuple t, b) An example relation D for representing students, c) the similarity of D tuples to t

needs to compute *term* and *document frequencies* to be able to operate. In many applications, spending some time for setting up the similarity function is reasonable since we setup the function once and use it many times. However, this is not the case for our MC algorithm because for each selected possible world, we have to spend a significant time to setup the similarity function that is used once, only for the selected possible world.

In this section, we propose *CB* (Community Based), a novel similarity function which avoids this problem since it does not need any time to setup. In fact, setting up the function and computing the similarities happen at the same time. It gives more importance to the more discriminative attributes of the two tuples by introducing the novel concept of *community*. A community C of a relation D is defined as the largest subset of D's tuples that are similar, i.e. their similarity is more than a certain threshold, based on a non-empty subset of D's attributes. In CB, the similarity of two tuples depends on the size of the most specific community to which they belong. The smaller the size of the community, the more similar are the two tuples.

For computing the similarity of a tuple t' to a given tuple t, CB finds the smallest community to which t and t' belong, say community C, and returns $1 - \frac{\log|C|}{\log|D|}$ as the similarity score of t and t'. If no community involves both t and t' , then CB returns 0 as similarity score. Indeed, if two tuples do not belong to any common community, they are not similar at all.

Figure 4(b) shows an example relation D on the schema S_D*(gender, city, age-range)* for describing people. As an example, let us take community $C_1 = \{t_3, t_4, t_5, t_6, t_7, t_8\}$ that represents females and community $C_2 = \{t_6, t_7, t_8\}$ that contains all females who live in city 'A'. Figure 4(a) shows an example tuple t and Figure 4(c) shows the similarity of all tuples of relation D to t. For instance, tuple t_8 is similar to t in *gender* and *city* attributes. Therefore, the smallest community involving t and t_8 is the community of females who live in city 'A', which includes tuples t_6, t_7, and t_8. Notice that there is no need for attribute values to be equal, but their similarity should be greater than a predefined threshold. We may use any string similarity function for obtaining the similarity of individual attribute values.

CB is context-sensitive since the similarity of two tuples depends not only on their attribute values, but also on the other tuples of the relation to which they belong. An important property of CB is that the similarity value drops as the tuples with matching values grows. In other words, a match between tuples is a stronger match if there aren't many of them that are the same. For instance, a tuple that is matched to a single other tuple is more important than matching with 100 other tuples. Let us now formally define CB.

Definition 2. ***CB similarity function****: Let D be a relation on schema S_D and t be a tuple on schema S_t and $S_t \subseteq S_D$. Suppose F is a similarity function that computes the similarity between two attribute values from the same domain. Let $S_i \subseteq (S_D \cap S_t)$ be the set of attributes in which t is similar to $t_i \in D$. Let $D_i \subseteq D$ be the set of tuples that are similar to t in all attributes involved in S_i. The similarity score of t to t_i, denoted as $score(t, t_i)$, is defined as*

$$score(t, t_i) = \begin{cases} 1 - \frac{\log|D_i|}{\log|D|} & \text{if } |D_i| \neq 0 \\ 0 & \text{if } |D_i| = 0 \end{cases}$$

Algorithm 5 Computing similarity scores based on the CB similarity function

Input:
- tuple t
- database $D = \{t_1, \ldots, t_n\}$

Output: array $Score[n]$, where $Score[i]$ represents the similarity between t and t_i

1: define hash table H and $H \leftarrow \emptyset$
2: **for all** $t_i \in D$ **do**
3: compute $S_i \subseteq S_D$ as the set of attributes in which t_i is similar to t
// let $D_i \subseteq D$ be the set of tuples that are similar to t in all attributes involved in S_i
4: **if** H contains key S_i **then**
5: retrieve $|D_i|$ as the value for key S_i from hash table H
6: **else**
7: compute D_i and store key-value pair $\langle S_i, |D_i| \rangle$ in hash table H
8: **end if**
9: **if** $|D_i| \neq 0$ **then**
10: $Score[i] \leftarrow 1 - \frac{\log|D_i|}{\log|D|}$
11: **else**
12: $Score[i] \leftarrow 0$
13: **end if**
14: **end for**

The most similar tuple t_{max} to t is defined as

$$t_{max} = \arg\max_{t_i \in D}(score(t, t_i)).$$

CB Algorithm Algorithm 5 shows the pseudo code of the CB similarity function. It takes as input tuple t and a set of tuples D. It uses a function for measuring the similarity of two attribute values of the same domain. For every tuple in the database, say t_i, steps 3-13 compute its similarity score. Step 3 computes the subset S_i of attributes in which t and t_i are similar. Steps 4-8 either compute or use the already computed set D_i of tuples that are similar to t in all attributes of S_i. The algorithm uses a hash table for managing the key-value pairs $\langle S_i, |D_i| \rangle$. Steps 9-13 compute the similarity score based on the size of the set D_i.

Notice that for each subset S_i of attributes, we compute the set of tuples that are similar in the attributes of S_i, only once. For instance in Figure 4(c), once we compute the set of tuples that are similar to tuple t in attribute set {*age-range*}, we use it for computing both $score(t, t_1)$ and $score(t, t_{10})$. This greatly reduces the execution cost of the algorithm.

Cost analysis Let us analyze the execution time of Algorithm 5. In this algorithm, the first step is to compute for every attribute a_i of t, a set A_i including the tuples of D that are similar to t in a_i. Let the execution time of matching two attribute values be $g(w)$ where w is the maximum size of the values, and k be the number of attributes that are common in the schemas of t and D. Thus, this step takes $\theta(n \times g(w))$ time for every attribute and $\theta(k \times n \times g(w))$ time for all k attributes, where n is the number of tuples involved in the database D. In the next step, for every tuple $t_i \in D$, we need to obtain the intersection of all A_j sets that have t_i as member, say set S_i. Obtaining the intersection of S_i members takes $O((|S_i| - 1) \times (n - 1))$ time, which is equal to $O(k \times n)$ since $|S_i|$ is bounded to k.

We have at most 2^k different S_i sets and n tuples in D. Thus, executing this step for all tuples in the worst case takes $min(2^k, n) \times O(k \times n)$ time. Therefore, the algorithm takes $min(2^k, n) \times O(k \times n) + \theta(k \times n \times g(w))$ time. Usually the maximum attribute value size w can be considered as a constant and $2^k < n$, i.e. $k < \log n$, thus the worst case execution time of the algorithm can be written as $O(n^2 \times \log n)$. However, our experiments show that different S_i sets that occur in practice are far less than n. Thus, the average execution time of the algorithm is much better than its worst case performance.

5 PERFORMANCE EVALUATION

In this section, we study the effectiveness of our algorithms through experimentation over synthetic and real datasets. The rest of this section is organized as follows. We first describe our experimental setup. Then, we study the performance of the MPM and MC algorithms. Afterwards, we compare the performance of our similarity function with competing approaches. Finally, we summarize the performance results.

5.1 Experimental setup

We implemented our CB similarity function, CFA, CFA-MPME, CFA-MPMP, and MC algorithms in Java. We also implemented the MapReduce version of the MC algorithm in Java using Hadoop framework. We compare CB with Jaccard, Levenshtein (also known as edit-distance), SoftTFIDF [30] and a version of SoftTFIDF introduced in [31], which we denote as Flexi-SoftTFIDF. We used the SecondString Java package[1] for implementing these similarity functions. The SoftTFIDF and Flexi-SoftTFIDF functions use a similarity function for finding similar tokens inside attributes. Also, CB uses a similarity function for finding similar attributes. For this purpose, we used Jaro-Winkler similarity function [24] for all of these methods and we used the same similarity threshold for all of them. To the best of our knowledge, there is no other approach for computing most-probable matches other than expanding the possible worlds, i.e. the naïve approach. As a result, we consider the naïve approach as the only competitor to our approach.

The datasets that we use for our experiments are listed in Table 1. The Cora, Restaurant, and Census datasets have been frequently used in the literature to evaluate similarity functions, e.g. [32–34, 30, 35, 36]. The duplicate tuples of these datasets have been labeled. The Cora dataset includes bibliographical information about scientific papers in 12 different attributes from which we only use *author*, *title*, and *venue* attributes. The Restaurant dataset includes the *name*, *address*, *city*, and the *type* attributes of some restaurants. The Census dataset is a synthetic census-like data from which we only use the textual attributes, i.e. *first name*, *middle initial*, *last name*, *street*, *city*, and *state*. In order to be able to control characteristics of the dataset, we use the Synthetic dataset which we generate ourselves.

We use a number of wordlists[2] for the attribute values of the Synthetic database. To generate the database, we use one wordlist, say wordlist W, for each attribute and distribute the words in that wordlist over that attribute using a Gaussian distribution with a mean of $|W|/2$ and a deviation of $|W|/6$, where $|W|$ is the size of the wordlist W.

Table 1. Datasets used in experiments. Source is the place where the dataset was originally introduced in, and k the number of used attributes, n the number of tuples of the dataset

Name	Source	k	n
Cora	[32]	3	1,295
Restaurant	[33]	4	864
Census	[34]	6	3,000
Synthetic	-	10..1,500	10..2,000,000

To evaluate the performance of CFA, CFA-MPME, CFA-MPMP, and MC algorithms, we use a probabilistic version of the Synthetic database. To control the characteristic of the database, we introduce d_x, called the x-degree, as the maximum number of alternatives in an x-tuple, n as the number of tuples in the database, and k as the number of attributes in the database. To generate the database, we first generate the non-probabilistic Synthetic database as we explained above, then we add an attribute named *probability* to the database and convert the generated database into a probabilistic database as follows. We generate d as a uniform random number in $[1, d_x]$, and repeatedly pick d tuples at random and group them into an x-tuple. The alternative tuples of the generated x-tuple are supposed to represent the same entity. To emulate this scenario, we randomly choose one of the x-tuple's alternatives as the seed tuple, say t, and

[1] http://secondstring.sourceforge.net
[2] ftp://ftp.ox.ac.uk/pub/wordlists

for each of the other $d-1$ alternatives, say t', we do the following: we randomly choose $\lfloor k \times 0.8 \rfloor$ attributes of t' and change their values with corresponding attribute values of t. For the *probability* attribute of the xtuple's alternatives, we use a discrete Gaussian distribution[3] with a mean of 0 and a deviation of 0.2. We repeat the process of generating x-tuples until we group all tuples in valid x-tuples. We use the default value $d_x = 10$. We generate the uncertain entity needed for experiments, in the same way that we generate a valid x-tuple and always with 3 alternatives.

As a context-free similarity function, we use the Jaro-Winkler similarity function [24] to evaluate the CFA algorithms. When comparing the MC algorithm with the naïve approach, we use CB similarity function in the naïve approach. Moreover, to study the performance of different context-sensitive similarity functions in the MC algorithm, we use SoftTFIDF, Flexi-SoftTFIDF, and CB similarity functions.

Method and performance metrics. To evaluate the performance of the similarity functions, we tested them over both synthetic (i.e. Census) and real datasets (i.e. Cora and Restaurant), thus covering all practical cases. We use success-rate for comparing the efficiency of different similarity functions. Our method for measuring the success-rate is described as follows. We repeatedly pick a tuple t from dataset D. If t has at least one duplicate tuple in D, except itself, we remove t from D and give the task of finding the most similar tuple in the dataset to t, to the similarity function. If the similarity function returns the duplicate tuple of t, we denote the search as a successful search. We repeat this process for all of the tuples of the dataset. The fraction of times the search is successful denoted as *success-rate* of the similarity function for dataset D.

We also perform a self-join on the test datasets to measure the performance of different similarity functions. The self-join operation over the dataset D returns all tuple pairs in D, say $(t, t'), t \neq t'$, whose similarity is more than a certain specified threshold θ. To summarize the test results, we use the maximum F1 score. The F1 score for a threshold is defined as

$$F1 = \frac{2 \times precision \times recall}{precision + recall}$$

where precision is the percentage of returned tuple pairs that are duplicates, and recall is the percentage of duplicate tuple pairs that are returned by the self-join operation. The maximum F1 score is simply the maximum value of F1 that is obtained for different threshold values.

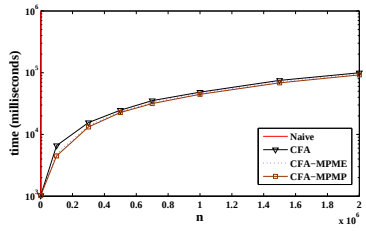


Fig. 5. Response times of Naïve and CFA algorithms

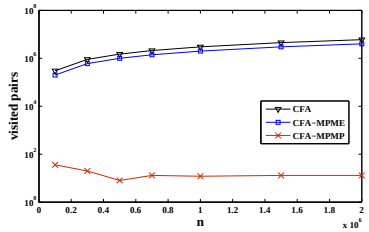


Fig. 6. The number of visited pairs vs. n in CFA algorithms

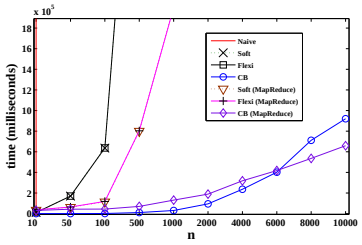


Fig. 7. Response times of naive and different implementations of MC

To evaluate the scalability of different similarity functions, we measure their *response time*, which is the time to find the most similar tuple to a given tuple. Some approaches such as SoftTFIDF and Flexi-SoftTFIDF need a time to setup the process. We denote this time as the *setup time* of these functions.

To evaluate the performance of the CFA and MC algorithms, we measure their response time and compare it with the naïve approach. Moreover, to compare the performance of the CFA algorithm with its two imrpoved vesrsions (i.e. CFA-MPME and CFA-MPME), we measure the numebr of tuple pairs which are visited by these algorithms.

[3] Indeed, we use a continues Guassian distribution with a mean of 0 and a deviation 0.2, say $P(x)$, then we divide the range [-0.6..0.6] into d equal ranges, say $r_1, \ldots, r_d$, and for each range r_i compute the value of $P(x) \in r_i$.

We conducted our single-machine experiments on a Windows 7 machine with Intel Xeon 3.3 GHz CPU and 32 GB memory. For the MapReduce implementation, we used the Sara Hadoop cluster[4] which consists of 20 machines each with Dual core 2.6 GHz CPU and 16 GB memory.

5.2 Results

Performance of CFA and MC algorithms In this section, we study the performance of CFA, CFA-MPME, CFA-MPMP, and MC algorithms. We use the synthetic database as our test database. In our experiments, we set the δ parameter of the MC algorithm to 0.1 (see Section 4).

With n increasing up to 2,000,000, Figure 5 compares the response time of the naïve approach with that of CFA and its two improved versions. This figure shows that the response time of the naïve approach increases exponentially with n but that of CFA increases very smoothly. The performance gain of CFA-MPME and CFA-MPMP over CFA is not significant because the time needed for sorting the tuple pairs dominated the time needed for visiting the pairs. In order to compare the performance gain of CFA-MPME and CFA-MPMP over CFA, Figure 6 shows the number of visited tuple pairs in these algorithms. We observe that while CFA-MPMP significantly outperforms CFA, its number of visited tuple pairs is almost constant and independent from n. Moreover, although the performance gain of CFA-MPME is far less than that of CFA-MPMP, it is still noticeable.

We conducted experiments to study the effectiveness of different context-sensitive similarity functions in the MC algorithm. Figure 7 shows the response times of the naïve approach and two implementations of the MC algorithm, i.e. on a single machine and on the MapReduce framework, using different similarity functions. Overall, both on a single machine and using the MapReduce framework, CB significantly outperforms SoftTFIDF and Flexi-SoftTFIDF. The response times of SoftTFIDF and Flexi-SoftTFIDF increases dramatically with increasing n, while that of CB increases smoothly. Figure 7 also shows that the MapReduce implementation always outperforms the single-machine implementation in SoftTFIDF and Flexi-SoftTFIDF. However, in CB, the MapReduce implementation outperforms the single-machine implementation for $n > 6,000$. This is because the overhead of the MapReduce framework is more than its gain for small values of n. Notice that the response times of SoftTFIDF and Flexi-SoftTFIDF are so close that cannot be differentiated in the figure.

Performance of CB In this section, we compare the performance of the CB similarity function with other competing similarity functions.

Figures 8 and 9 respectively compare the success-rate and maximum F1 of CB, SoftTFIDF, Flexi-SoftTFIDF, Jaccard, and Levenshtein similarity functions over the Cora, Restaurant, and Census datasets. We observe that CB always performs a little less than the best similarity function and never is the worst similarity function.

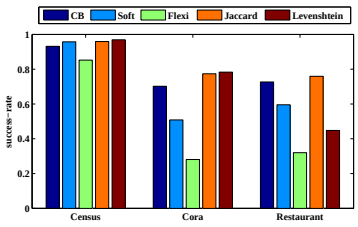


Fig. 8. Success-rate of similarity functions over different datasets

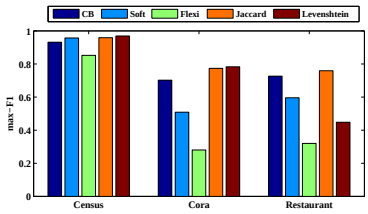


Fig. 9. Max-F1 of similarity functions over different datasets

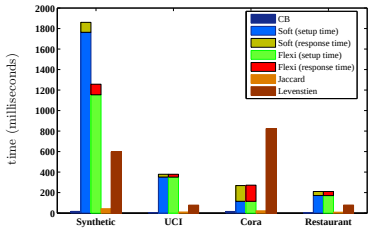


Fig. 10. Average setup time and response time of different similarity functions

Figure 10 shows the average setup time and average response time of performing a search for different similarity functions over different datasets. CB outperforms all similarity functions over all datasets. The performance gain is

[4] http://www.sara.nl/project/hadoop

significant for the context-sensitive similarity functions (i.e. SoftTFIDF and SoftTFIDF) over all datasets. This is one of the reasons that SoftTFIDF and Flexi-SoftTFIDF perform inefficiently in the MC algorithm.

We also conducted experiments to study how the response time of different similarity functions evolves with increasing n and also k (i.e. the number of attributes). Figure 11 shows the response time of a search over Synthetic dataset with increasing n up to 500,000 and k set to 10. This figure shows that CB not only outperforms the other similarity functions but also scales far better than them with the number of tuples in the dataset. Figure 12 shows the response time of a search over Synthetic dataset with increasing k up to 1,500 and n set to 3000. This figure shows that except Jaccard, CB outperforms all other similarity functions. We also observe that CB scales very well with the number of attributes. Notice that in these figures, the response times of SoftTFIDF and Flexi-SoftTFIDF are so close that cannot be differentiated from each other.

These results reveal another reason for outperformance of CB over other similarity functions in the MC algorithm. There are two main reasons for the good response time of CB compared to the other methods. The first reason is that CB uses an efficient string similarity function (i.e. Jaro-Winkler in our experiments) for computing the similarity between individual attributes. The second reason is that while all other methods compute the score of tuples one by one, CB reuses the computed score of a tuple for other tuples that are in the same situation (i.e. are similar in the same subset of attributes). This greatly improves the response time.

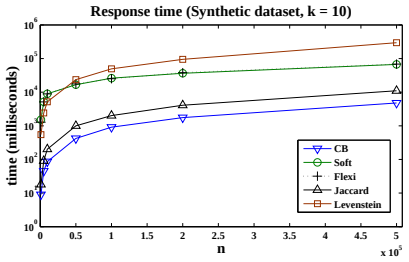


Fig. 11. Response time of different similarity functions vs. number of tuples

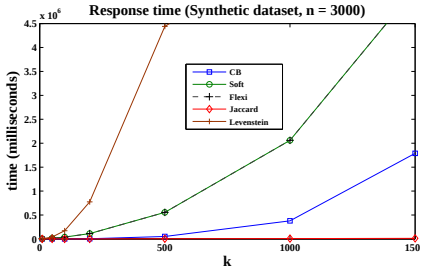


Fig. 12. Response time of different similarity functions vs. number of attributes

6 Related work

In the literature, considerable attention has been devoted to the problem of identifying different representations of the same real world entity over certain data (refer to [15] for a survey). The problem has been presented under various terms such as *entity resolution*, *identity resolution*, *duplicate detection*, etc. Recently, there have been some proposals dealing with entity resolution over probabilistic data [20, 21]. The proposals in [20, 21] consider the duplicate detection of an uncertain relation and hence generate a clustering of the relation's tuples. In these proposals, one of the main problems is the merging of the tuples that fall in the same cluster. In contrast, we consider an entity resolution scenario where the aim is to resolve a given uncertain entity against an uncertain database, by considering both similarity and probability of database tuples and the given entity, and using the possible worlds semantics.

Among the prior work in database literature, nearest neighbor query processing, and top-k query processing over uncertain data are relevant to ours.

While there are a number of proposals, e.g. [37, 38, 12], that deal with nearest neighbor queries over uncertain data, to the best of our knowledge, all of them model uncertain data by means of probability density functions (pdfs), which thus make their problem very different than ours.

There also have been some proposals dealing with uncertain top-k query processing (e.g., [8–11]). The work in [9] extends the semantics of top-k queries from certain to uncertain databases and enumerate the possible worlds space of the uncertain database to compute the query results. We inspired by the semantics defined in [9] for defining the new semantics of entity resolution over probabilistic data. The proposal in [10] avoids enumerating the possible worlds

space by using a dynamic programming approach for efficiently processing top-k queries on uncertain databases. Overall the top-k proposals rely on a ranking function, which assigns a fixed unique rank to every uncertain tuple in the query result, but our problem setting is different in both context-free and context-sensitive similarity cases. In the case of context-free similarity function, we are faced with multiple alternatives of the given uncertain entity, which results in multiple similarity scores between the uncertain entity and a tuple in the database. In the case of using a context-sensitive similarity function, we are not only faced with multiple similarity scores, but also the rank of a tuple may change in different possible worlds.

There is an extensive literature related to similarity functions that compare individual attributes stored as strings (refer to [30] and [39] for surveys). There also are a number of proposals for comparing tuples (refer to [40] for a recent survey). They mainly rely on two main approaches for comparing tuples. The first approach is to treat a tuple as a long field and apply one of the string similarity functions. This approach can be applied to all of the string similarity functions (e.g., TFIDF, SoftTFIDF, and Monge-Elkan). The second approach is to compute the similarity values between individual attributes of the two tuples and combine these values to obtain a whole similarity value. In [41], the weighted similarity is computed by assigning static weights to attributes based on their importance. The proposed technique in [31] modifies the normalization step of the well-known TFIDF similarity function to dynamically adjust the attributes' weights based on their relative importance. [29] creates a similarity function based on ranked list merging. The basic idea is that if we compare only one attribute from the tuple, the matching algorithm can easily find the best matches and rank them according to their similarity, putting the best matches first [15]. Our similarity function differs from these proposals since ours is built on the novel concept of communities, which neither explicitly nor implicitly has been considered by prior proposals. An important advantage of our similarity function is that it does not need any extra pass over the database for setup. This makes our similarity function more suitable for the ERPD problem.

7 Conclusion

In this paper, we considered the problem of Entity Resolution for Probabilistic Data (ERPD), which is crucial for many applications that process probabilistic data. We adapted the possible worlds semantics of probabilistic data to define the novel concepts of most-probable match pair and most-probable match entity as the outcomes of ERPD. Then, we proposed MPM, a PTIME algorithm, which is applicable to the context-free similarity functions. The MPM algorithm is not applicable for context-sensitive similarity functions. Thus, we used Monte Carlo algorithm for approximating the outcome of ERPD. Existing context-sensitive similarity functions need a setup time for passing over the database to compute some statistical features of data. This shortcoming makes these functions inefficient for our proposed Monte Carlo algorithm. Thus, with the aim of having a context-sensitive similarity function with no setup time, we proposed the CB similarity function. We validated CB over both synthetic and real datasets. Our experiments show that CB significantly outperforms other similarity functions in terms of response time and success rate, and in using it in our proposed Monte Carlo algorithm. To further improve the efficiency our proposed Monte Carlo algorithm, we proposed a parallel version of it using the MapReduce framework. Our experiments show that the parallel version of Monte Carlo outperforms the single-machine Monte Carlo algorithm.

References

1. Magnani, M., Montesi, D.: Uncertainty in data integration: current approaches and open problems. In: Proc. of MUD. (2007)
2. Ayat, N., Afsarmanesh, H., Akbarinia, R., Valduriez, P.: An uncertain data integration system. In: Proc. of ODBASE. (2012)
3. Deshpande, A., Guestrin, C., Madden, S., Hellerstein, J.M., Hong, W.: Model-driven data acquisition in sensor networks. In: Proc. of VLDB. (2004)
4. Jayram, T.S., Krishnamurthy, R., Raghavan, S., Vaithyanathan, S., Zhu, H.: Avatar information extraction system. IEEE Data Eng. Bull. **29**(1) (2006)
5. Sarma, A.D., Benjelloun, O., Halevy, A.Y., Widom, J.: Working models for uncertain data. In: Proc. of ICDE. (2006)
6. Benjelloun, O., Sarma, A.D., Halevy, A.Y., Widom, J.: Uldbs: Databases with uncertainty and lineage. In: Proc. of VLDB. (2006)
7. Atallah, M.J., Qi, Y.: Computing all skyline probabilities for uncertain data. In: Proc. of PODS. (2009)

8. Cormode, G., Li, F., Yi, K.: Semantics of ranking queries for probabilistic data and expected ranks. In: Proc. of ICDE. (2009)
9. Soliman, M.A., Ilyas, I.F., Chang, K.C.C.: Top-k query processing in uncertain databases. In: Proc. of ICDE. (2007)
10. Yi, K., Li, F., Kollios, G., Srivastava, D.: Efficient processing of top-k queries in uncertain databases with x-relations. TKDE **20**(12) (2008)
11. Re, C., Dalvi, N.N., Suciu, D.: Efficient top-k query evaluation on probabilistic data. In: Proc. of ICDE. (2007)
12. Yuen, S.M., Tao, Y., Xiao, X., Pei, J., Zhang, D.: Superseding nearest neighbor search on uncertain spatial databases. TKDE **22**(7) (2010)
13. Abiteboul, S., Kimelfeld, B., Sagiv, Y., Senellart, P.: On the expressiveness of probabilistic xml models. VLDB J. **18**(5) (2009)
14. Antova, L., Jansen, T., Koch, C., Olteanu, D.: Fast and simple relational processing of uncertain data. In: Proc. of ICDE. (2008)
15. Elmagarmid, A.K., Ipeirotis, P.G., Verykios, V.S.: Duplicate record detection: A survey. TKDE **19**(1) (2007)
16. Peng, L., Diao, Y., Liu, A.: Optimizing probabilistic query processing on continuous uncertain data. PVLDB **4**(11) (2011)
17. Benjelloun, O., Sarma, A.D., Halevy, A.Y., Widom, J.: Uldbs: Databases with uncertainty and lineage. In: Proc. of VLDB. (2006)
18. Cheng, R., Chen, J., Xie, X.: Cleaning uncertain data with quality guarantees. PVLDB **1**(1) (2008)
19. Suciu, D., Connolly, A., Howe, B.: Embracing uncertainty in large-scale computational astrophysics. In: MUD. (2009)
20. Menestrina, D., Benjelloun, O., Garcia-Molina, H.: Generic entity resolution with data confidences. In: Proc. of CleanDB. (2006)
21. Panse, F., van Keulen, M., de Keijzer, A., Ritter, N.: Duplicate detection in probabilistic data. In: Proc. of ICDE Workshops. (2010)
22. Agrawal, P., Benjelloun, O., Sarma, A.D., Hayworth, C., Nabar, S.U., Sugihara, T., Widom, J.: Trio: A system for data, uncertainty, and lineage. In: Proc. of VLDB. (2006)
23. Chaudhuri, S., Ganjam, K., Ganti, V., Motwani, R.: Robust and efficient fuzzy match for online data cleaning. In: Proc. of SIGMOD Conference. (2003)
24. Winkler, W.E., Thibaudeau, Y.: An Application Of The Fellegi-Sunter Model Of Record Linkage To The 1990 U.S. Decennial Census. In: U.S. Decennial CensusŤ. Technical report, US Bureau of the Census. (1987)
25. Monge, A.E., Elkan, C.: The field matching problem: Algorithms and applications. In: Proc. of KDD. (1996)
26. Monge, A.E., Elkan, C.: An efficient domain-independent algorithm for detecting approximately duplicate database records. In: Proc. of DMKD. (1997)
27. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady **10**(8) (1966)
28. Cohen, W.W.: Integration of heterogeneous databases without common domains using queries based on textual similarity. In: Proc. of SIGMOD Conference. (1998)
29. Guha, S., Koudas, N., Marathe, A., Srivastava, D.: Merging the results of approximate match operations. In: Prof. of VLDB. (2004)
30. Bilenko, M., Mooney, R.J., Cohen, W.W., Ravikumar, P.D., Fienberg, S.E.: Adaptive name matching in information integration. IEEE Intelligent Systems **18**(5) (2003)
31. Koudas, N., Marathe, A., Srivastava, D.: Flexible string matching against large databases in practice. In: Proc. of VLDB. (2004)
32. McCallum, A., Nigam, K., Ungar, L.H.: Efficient clustering of high-dimensional data sets with application to reference matching. In: Proc. of KDD. (2000)
33. Tejada, S., Knoblock, C.A., Minton, S.: Learning object identification rules for information integration. Inf. Syst. **26**(8) (2001)
34. Hernández, M.A., Stolfo, S.J.: Real-world data is dirty: Data cleansing and the merge/purge problem. Data Min. Knowl. Discov. **2**(1) (1998)
35. Dong, X., Halevy, A.Y., Madhavan, J.: Reference reconciliation in complex information spaces. In: Proc. of SIGMOD. (2005)
36. Bilenko, M., Mooney, R.J.: Adaptive duplicate detection using learnable string similarity measures. In: Proc. of KDD. (2003)
37. Kriegel, H.P., Kunath, P., Renz, M.: Probabilistic nearest-neighbor query on uncertain objects. In: DASFAA. (2007)
38. Trajcevski, G., Tamassia, R., Ding, H., Scheuermann, P., Cruz, I.F.: Continuous probabilistic nearest-neighbor queries for uncertain trajectories. In: Proc. of EDBT. (2009)
39. Cohen, W.W., Ravikumar, P.D., Fienberg, S.E.: A comparison of string distance metrics for name-matching tasks. In: Proc. of IIWeb. (2003)
40. Dorneles, C.F., Gonçalves, R., dos Santos Mello, R.: Approximate data instance matching: a survey. Knowl. Inf. Syst. **27**(1) (2011)
41. Dey, D., Sarkar, S., De, P.: Entity matching in heterogeneous databases: a distance-based decision model. In: Proc. of 31st Hawaii International Conference on System Sciences. (1998)