Entity Resolution for Uncertain Data

Naser Ayat, Reza Akbarinia, Hamideh Afsarmanesh, Patrick Valduriez

Entity Resolution for Uncertain Data

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

[#]Informatics Institute, University of Amsterdam, Amsterdam, Netherlands

[1]s.n.ayat@uva.nl, [3]h.afsarmanesh@uva.nl

[*]INRIA and LIRMM, Montpellier, France

[2,4]{firstname.lastname@inria.fr}

Abstract

Entity resolution (ER), also known as duplicate detection or record matching, is the problem of identifying the tuples that represent the same real world entity. In this paper, we address the problem of ER for uncertain data, which we call ERUD. We propose two different approaches for the ERUD problem based on 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 algorithm for context-sensitive similarity functions. Existing context-sensitive similarity functions need at least one pass over the database to compute some statistical features of data, which makes it very inefficient for our Monte Carlo algorithm. Thus, we propose a novel context-sensitive similarity function that makes our Monte Carlo algorithm more efficient. To further improve the efficiency of our proposed Monte Carlo algorithm, we propose a parallel version of it using the MapReduce framework. We validated our algorithms through experiments over both synthetic and real datasets. Our performance evaluation shows the effectiveness of our algorithms in terms of success rate and response time.

1 INTRODUCTION

In recent years, we have been witnessing much interest in uncertain data management in many application areas such as data integration [21], sensor networks [11, 17], information extraction [16], etc. Much research effort has been devoted to several aspects of uncertain data management, including data modeling [28, 5], skyline queries [4, 26], top-k queries [10, 29, 31, 27], nearest neighbor search [33], spatial queries [32], XML documents [1, 18], 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 [3, 28, 5].

An important problem that arises in many applications such as information integration is that of Entity Resolution (ER) [14]. 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 [14] for a survey), but it has not been deeply investigated for uncertain data. Before discussing existing solutions, let us first motivate the need for ER in uncertain data (which we call ERUD), with two examples from environmental and anti-criminal domains.

Example 1: Protecting endangered animals. In order to protect endangered animals, a national park uses a video surveillance system for tracking these animals. The system consists of a number of nodes in different geographical locations, and a central node. Every local node consists of a video camera and a local video processing system that has been trained with sample images of animals. The task of local nodes is to permanently watch the environment; look for the target endangered animals in the environment; and report the presence of these animals to the central node. The source of uncertainty that arises is twofold. First, the outcome of the object detector that matches an image in the video with one of the target animals (e.g. Felzenszwalb object detector[1] and the Pittsburgh Pattern Recognition[2]) is uncertain. Second, the extracted attributes of the detected animal (referred to as features in image processing literature) is not certain due to the special environmental conditions such as rain, low light, and the quality of the camera. Thus, every successful hit of an animal in a node is reported as an uncertain entity to the central node. The central node stores the uncertain entity together with the timestamp and the reported location in an uncertain database. Park rangers can better protect the endangered animals by querying this database. A sample query on this database is as follows. Given the features of an animal reported by a local node, return all locations where such animal has been observed. Answering this query involves matching the specified uncertain entity with the uncertain entities stored in the database.

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 instance, the police cannot completely trust informants and witnesses. Thus, confidence values can be attached to them to show their likelihood of truth according to the confidence on the sources. These uncertain 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. When a crime occurs, detectives gather data about the perpetrator and represent these data in the form of an uncertain entity, say e. Besides regular SQL queries on the Suspects relation, detectives are also interested in finding instances and entities that are similar to e to get more information about the perpetrator represented by e. Note 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 confidence values completely and return the most similar tuples as the solution. Furthermore, the semantics of the solution for the ERUD problem has not been clearly defined in the literature. To the best of our knowledge, the works presented in [22] and [25] are the only proposals for dealing with the ERUD problem. The proposal in [22] ignores the confidence values and the proposal in [25] uses the confidence values only for normalizing the similarity of the

[1] http://people.cs.uchicago.edu/ pff/latent

[2] http://www.pittpatt.com

uncertain entities, i.e. mapping it to the range [0,1]. Inspired by the literature on uncertain data management, the goal of this paper is to adopt the well-known possible worlds semantics for defining the semantics for the ERUD problem and propose efficient algorithms for computing it.

We formulate the ERUD problem as follows. Let *e* be an uncertain entity represented by multiple possible alternatives, i.e. tuples, each with a confidence value. Let *D* be an uncertain database composed of a set of uncertain entities. Then, given *e*, *D*, and a similarity function *S*, the problem is to find the most similar entity in *D* to *e*, as well as the tuple in *D* that is closest to one of the *e*'s alternatives. Note that with uncertain data, we must redefine the concept of matching entities, which is different from certain databases where each entity is represented by only one alternative. This problem is challenging for the following reasons. First, we must take into account two parameters for matching the entities: the similarity and the confidence value. Second, uncertainty in *e* results in different similarity values between *e* and the tuples of *D*. Third, when we use *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 naive solution for ERUD 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 ERUD problem and propose a PTIME algorithm for one of the two classes of similarity functions. For the other class of similarity functions, we use Monte Carlo randomization for approximating the answer. To the best of our knowledge, this is the first study of the ERUD problem that adopts the possible world semantics and the first efficient algorithm for implementing it. Our contributions are summarized as follows:

- We adapt the possible worlds semantics of uncertain data to define the problem of ERUD based on novel concepts of most-probable match pair and most-probable match entity.
- We propose a PTIME algorithm for the ERUD problem. This algorithm is applicable to one of the two existing classes of the similarity 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 similarity functions, which makes them very inefficient for the Monte Carlo algorithm. We propose a new 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 conduct an extensive experimental study to evaluate our approach for ERUD over both real and synthesis datasets.

The rest of the paper is organized as follows. In Section 2, we present our uncertain data model, and define the problem we address. In Section 3, we propose our solution for the ERUD problem. In section 4, we present our solutions for improving the performance of the Monte Carlo algorithm. 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	Alternative	Age	Height	Weight	Eye color	Confidence
e	**t_1**	35-40	175-180	90+	Gray	0.4
	t_2	25-30	185-190	-	Blue	0.5

(a)

Suspects	Alternative	FName	LName	Age	Height	Weight	Gender	Eye color	Photo	Confidence
e_1	**t_3**	F1	L1	38	178	70	M	Gray	Ph1	0.6
	t_4	F1	L1	36	168	80	M	Hazel	Ph2	0.4
e_2	**t_5**	F2	L2	36	180	75	F	Blue	Ph3	0.3
	t_6	F2	L2	30	185	70	F	Gray	Ph4	0.5

(b)

World W	p(W)
$\{t_3\}$	$0.6 \times (1 - 0.3 - 0.5) = 0.12$
$\{t_4\}$	$0.4 \times (1 - 0.3 - 0.5) = 0.08$
$\{t_3, t_5\}$	$0.6 \times 0.3 = 0.18$
$\{t_3, t_6\}$	$0.6 \times 0.5 = 0.3$
$\{t_4, t_5\}$	$0.4 \times 0.3 = 0.12$
$\{t_4, t_6\}$	$0.4 \times 0.5 = 0.2$

(c)

Figure 1: a) Example of an entity in 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 uncertain data model. Then, we define the problem which we address.

2.1 Uncertainty Model

Entity model

In the traditional relational model, an entity is represented as a tuple by assigning a value to each attribute of the relation. For example, consider schema *S(Age, Height, Weight, Hair-color, Eye-color)* for representing people and entity e_c*(35, 175, 80, 'Black', 'Gray')* for representing a specific person. This model is suitable in situations where we are certain about the attribute values of an entity. We need a more powerful model in situations that we need to represent an uncertain entity. For this, we adopt the popular x-tuple uncertain entity model [2]. In x-tuple uncertainty model, an entity is represented by several possible certain entities, called alternatives, each associated with a confidence value so that the sum of the confidence values is less than or equal to 1. At each time, at most one alternative can exist. As an example of x-tuple, consider the entity that is shown in Figure 1(a). Note that either t_1 or t_2 is valid at a given time. The attributes age, height, and weight can take values in an interval.

Database model

We adopt a popular uncertain database model in this paper. This model, denoted as *x-relation* model [2], uses the x-tuple uncertainty model, which we described above as a model for representing uncertain entities. In the x-relation model, the uncertain database D consists of a set of pairwise disjoint x-tuple entities. At most one alternative of an entity can exist in a possible world and different entities can exist independently from each other. Formally,

$$D = \{e^1, \ldots, e^n\}, e^i = \{t^i_1, \ldots, t^i_{|e_i|}\}, e^i \neq \emptyset, e^i \cap e^j = \emptyset, i, j \in [1..n].$$

Let T be the set of all tuples appearing in D, i.e. $T = \bigcup_{i \in [1..n]} e^i$, and $p(t)$ be the probability of the occurrence of t, and W be any subset of T. The probability of W occurring is $p(W) = \prod_{i=1}^{n} p(e^i)$, where for any $e^i \in D$, $p(e^i)$ is defined as:

$$p(e^i) = \begin{cases} p(t) & \text{if } W \cap e^i = t \\ 1 - \sum\limits_{t \in e^i} p(t) & \text{if } W \cap e^i = \emptyset \\ 0 & \text{otherwise} \end{cases}$$

If $p(W) > 0$, we say W is a *possible world* of D. The set of all possible worlds denoted as possible worlds space of uncertain database D. We refer to D as x-relation. Figures 1(b) and 1(c) show an example x-relation, named *Suspects*, and all of its corresponding possible worlds, respectively. In the Suspects relation, t_3 and t_4 are e_1's alternatives and t_5 and t_6 are e_2's alternatives.

2.2 Problem Statement

In a *certain* database, the problem of entity resolution is defined as follows [7]: given a tuple t and a database D, find $t_{max} \in D$ such that t_{max} has the maximum similarity to t. While the problem of finding t_{max} is semantically clear in certain data, its semantics is not clear in uncertain data since we have to consider two parameters: similarity and probability. The interaction between the concepts *most similar* and *most probable* makes different definitions possible. The followings are two of the main definitions:

- The most similar tuple in the most probable world.
- The tuple that has the highest probability to be the most similar in all possible worlds.

In this paper, we deal with the second definition since the first definition ignores the fact that the aggregated probability of a tuple, other than the most similar tuple in the most probable world, may exceed the probability of the most probable world.

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 $Pr_{first}(t)$, 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 Pr_{first} 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_i, t_j), $t_i \in e, t_j \in D$. We refer to this pair as *most-probable match pair* and we use the first element of the pair for entity e and the second element for relation D. In addition to the most-probable match pair concept, uncertain data enables us to define another new concept, which does not make sense in *certain* data: *most-probable match entity*. The most-probable match entity e is an entity in D that has the highest probability of being most similar to e. Also to

Rank	Pair	Pr
1	(t_1, t_5)	0.12
2	**(t_1, t_3)**	0.168
3	(t_1, t_6)	0.08
4	(t_2, t_5)	0.15
5	(t_2, t_4)	0.14
6	(t_2, t_6)	0.15
7	(t_1, t_4)	0.032
8	(t_2, t_3)	0.06

(a)

Entity	Pr
e_1	0.168+0.032+0.06+0.14 = 0.4
e_2	0.12+0.08+0.15+0.15 = 0.5

(b)

Figure 2: Example of *most-probable matches* in the *Suspects* relation

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 *Suspects* relation and the entity e from the motivating example 2 in the previous section and suppose there is a similarity function which ranks the tuple pairs based on their similarity with each other. Figure 2 shows the result of computing the probability of being the most similar pair for each of eight possible tuple pairs and the most similar entity for the two entities e_1 and e_2. This figure shows that (t_1, t_3) is the most-probable match pair and entity e_2 is the most-probable match entity.

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

Definition 1 ***Most-probable matches****:*
Let $D = \{e^1, \ldots, e^n\}$, $e^i = \{t^i_1, \ldots, t^i_{|e_i|}\}, i \in [1..n]$ be an uncertain x-relation with possible worlds $PW(D)$ on schema S_D. Let e be an uncertain x-tuple entity on schema S_e and $S_e \subseteq S_D$. Let $W = e \times PW(D)$ be the possible worlds space of e and D. Let $X = \{x^1, \ldots, x^m\}$ be the set of pairs $x^i = (t_j, t_k), x^i \in w, w \in W, t_j \in e, t_k \in D$, where in each pair, t_k is the most similar tuple (t_{max}) to t_j in a non-empty set of possible worlds $PW(x^i) \subseteq W$ according to the S similarity function. Then, we define:

- *The* ***most-probable match pair*** *as:* $\arg\max_{x^i} \left(\sum_{w \in PW(x^i)} Pr(w) \right)$
- *The* ***most-probable match entity*** *as:* $\arg\max_{e^k,(t_j,t^k_l) \in x^i} \left(\sum_{w \in PW(x^i)} Pr(w) \right)$

As mentioned earlier, we refer to these concepts as *most-probable matches*. Given uncertain x-tuple entity e and uncertain x-relation D, our goal is to efficiently find the most-probable matches.

3 Computing Most-probable Matches

The entity resolution approach strongly depends on the similarity function that computes the similarity between the given entity and the tuples of the database. In the literature, several similarity functions have been introduced (see [6] and [9] for surveys). They 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 [30], Monge-Elkan [23, 24] and Levenshtein [20] 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 [8] and Ranked-List-Merging [15] are examples of context-sensitive similarity functions. While previous research has shown that there is no similarity function suitable for all domains [6], the context-sensitive functions are shown to be more promising than context-free functions for data similarity computation [6, 9].

In this section, we consider both classes of similarity functions for the ERUD problem. We first assume a context free similarity function, and propose a solution for computing the most-probable matches over an uncertain x-relation. Then, for the case of context-sensitive similarity functions, we propose a Monte Carlo approximation.

3.1 Context-free Similarity Function

In the case of context-free similarity function, the similarity score of a pair (tuple, tuple) remains constant in all possible worlds where the tuple appears. We rely on this fact to efficiently compute for each tuple pair the aggregate probability of being the most similar. Consider a pair (t, t') such that t is an alternative of the entity e and t' an alternative of the database x-tuples. The core idea of our approach is that in the case of context-free similarity function, the probability that the pair (t, t') be the most similar, is equal to the probability that (t, t') exists in the database and all pairs with higher similarities do not exist.

Algorithm 1 describes the details of our method for computing the most-probable matches. This algorithm, denoted as MPM algorithm, takes as input an uncertain x-relation database D, uncertain x-tuple entity e, and context-free similarity function Sim. Steps 4-9 are repeated for every alternative of e, say e_i. Step 4 computes the similarity score of e_i to every tuple of D; sorts the tuples based on their similarity scores; and stores the result in list L. For every tuple t_k, steps 6-10 compute the probability of the event that t_k appears as the most similar in the possible worlds of D. This event is the intersection of two independent events: e_i exists; and among tuples t_1 to t_k only t_k exists. For computing the probability of the latter event, the correlated tuples have to be grouped together. This is done by steps 6-9. Step 6 put the tuples t_1 to t_k in the set T_k and step 7 obtains the intersection of every x-tuple in D with the members of T_k. The resulted sets are in fact the correlated members of T_k, which are grouped together. Step 8 computes the group that includes t_k, say E group, and step 9 puts the other groups in set N. We are looking for possible worlds in which tuple t_k from group E exists and none of the members of the groups in N exist. In such possible worlds, pair (e_i, t_k) is the most probable. Step 10 computes the product of these probabilities to obtain the probability that the pair (e_i, t_k) appears at the first rank in the possible worlds of D. When the algorithm finishes computing the probabilities of all pairs, step 14 finds the most-probable matches. This step finds the most-probable match pair by finding the record in sets S with the maximum value for pr field. The most-probable match entity can be computed by aggregating the pr field for every entity $e^i \in D$ and then finding the entity with maximum aggregated pr value.

Let us analyze the execution time of MPM Algorithm as follows. Let n be the number of tuples involved in D, and m be the number of alternatives of e. Step 4 takes $O(n \times logn)$

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

```
    Input:
        - Uncertain x-relation database $D = \{e^1, \ldots, e^q\}$ from tuple domain $T = \bigcup_{i=1}^{q} e^i$
        - Uncertain entity $e = \{e^1, \ldots, e^m\}$
        - Context-free similarity function $Sim$
    Output: Most-probable matches
 1: define list $L$
 2: define set $S$ and $S \leftarrow \emptyset$
 3: for all $e_i \in e$ do
 4:     sort $D$ tuples based on $Sim(e_i, t_j)$, $t_j \in D$, in descending order and store
        the result in list $L$
 5:     for all $t_k \in L$ do
 6:         $T_k \leftarrow \{L[1], \ldots, L[k]\}$
 7:         $D_k \leftarrow \{e^i \mid e^i = e^j \cap T_k, e^j \in T, e^i \neq \emptyset\}$
 8:         $E \leftarrow \{e^i \mid e^i \in D_k, e^i \cap t_k \neq \emptyset\}$
 9:         $N \leftarrow D_k - E$
10:         $pr \leftarrow p(t_k) \times p(e_i) \times \prod_{e^j \in N} \left(1 - \sum_{t \in e^j} p(t)\right)$
11:         add record $(e_i, t_k, pr)$ to set $S$
12:     end for
13: end for
14: find the most-probable matches using set $S$ and return them
```

time for computing the similarity scores and $O(n \times logn)$ for sorting the scores. An efficient implementation of steps 6-9 takes $O(k + q)$ time, where q is the number of x-tuples in D. Thus, steps 5-9 take $\sum_{k=1}^{n} O(k + q)$ time, which is $O(n^2)$ since $q < n$. Therefore, steps 4-9 are done in $O(n^2)$. Since these steps are repeated m times (i.e. the number of alternatives of e), steps 3-9 take $O(m \times n^2)$ time. Step 10 takes $O(m \times n)$ time, which is dominated by $O(m \times n^2)$. Thus, the execution cost of the algorithm is $O(m \times n^2)$.

3.2 Context-sensitive Similarity Function

In this section, we first present a Monte Carlo approach for approximating the most-probable matches.

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 MPM algorithm for computing most-probable matches.

In the absence of an efficient exact algorithm for ERUD 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 pair that is the most similar. For each pair $\omega =$

(e_i, t_j), the probability $p(\omega)$ of being the most similar match, is approximated by $p'(\omega)$, which is the fraction of times that t_j was the most similar to e_i. The MC algorithm guarantees that after sampling N possible worlds, $p'(\omega)$ is in the interval $\left[p(\omega) - \frac{z\delta}{\sqrt{N}}, p(\omega) + \frac{z\delta}{\sqrt{N}}\right]$ 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.

We don't know δ a priori, but we can approximate it. To use MC algorithm, we have to first estimate the needed N. For this, we first simulate a small number N_0 of trial runs, thereby yielding $X_0^\omega, \ldots, X_{N_0}^\omega$ for each pair ω. We then compute $P_{N_0}^\omega = \frac{X_0^\omega + \ldots + X_{N_0}^\omega}{N_0}$ and $\delta_{N_0}^\omega = \sqrt{\frac{\sum_{i=1}^{N_0} (X_i^\omega - P_{N_0}^\omega)^2}{N_0 - 1}}$. Assuming that we need a final confidence interval with half-width ϵ, we compute N^ω as $N^\omega \approx \frac{z^2(\delta_{N_0}^\omega)^2}{\epsilon^2}$, and N as $N = max(N^\omega)$. Then, we have to sample N possible worlds to obtain the desired confidence.

4 Improving performance of MC Algorithm

In this section, we propose two solutions to improve the performance of the MC algorithm which we presented in previous section. First, we propose a new context-sensitive similarity function is appropriate for Monte Carlo 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.

Then, we propose a parallel version of the MC algorithm using the MapReduce framework.

4.1 CB Similarity Function

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 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. 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 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 $1/|C|$ 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 3(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 3(a) shows an example tuple t and Figure 3(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.

	Gender	**City**	**Age-range**
t	F	A	18-40

(a)

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

(b)

	Smallest community	**Similarity**
$\mathbf{t_1}$	$\{t_1,t_3,t_6,t_7,t_{10}\}$	1/5
$\mathbf{t_2}$	$\emptyset$	0
$\mathbf{t_3}$	$\{t_3,t_6,t_7\}$	1/3
$\mathbf{t_4}$	$\{t_3,t_4,t_5,t_6,t_7,t_8\}$	1/6
$\mathbf{t_5}$	$\{t_3,t_4,t_5,t_6,t_7,t_8\}$	1/6
$\mathbf{t_6}$	$\{t_6,t_7\}$	1/2
$\mathbf{t_7}$	$\{t_6,t_7\}$	1/2
$\mathbf{t_8}$	$\{t_6,t_7,t_8\}$	1/3
$\mathbf{t_9}$	$\{t_6,t_7,t_8,t_9\}$	1/4
$\mathbf{t_{10}}$	$\{t_1,t_3,t_6,t_7,t_{10}\}$	1/5

(c)

Figure 3: a) An example tuple t, b) An example relation D for representing students, c) the similarity of D tuples to t

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. Below, we formally define it.

Definition 2 ***CB similarity function**: Let D be a relation on schema S_D and e be a tuple on schema S_e and $S_e \subseteq S_D$. Suppose S is a similarity function that computes the similarity between two attribute values from the same domain. Let $A' \subseteq A$ be the set of attributes in which e is similar to $t \in D$. Let $D' \subseteq D$ be the set of tuples that are similar to e in all attributes involved in A'. The similarity score of e to t, denoted as $score(e, t)$, is defined as*

$$score(e,t) = \begin{cases} \frac{1}{|D'|} & if\ |D'| \neq 0 \\ 0 & if\ |D'| = 0 \end{cases}$$

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

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

In the rest of this paper, unless otherwise stated, we mean CB whenever we refer to a similarity function .

4.1.1 CB Algorithm

Algorithm 2 shows the pseudo code of the CB similarity function. It takes as input tuple e and a set of tuples D. It uses a function for estimating the similarity of two attribute values of the same domain. For every tuple in the database, say t_i, steps 3-9 compute its similarity score. Step 3 computes the subset A' of attributes in which e and t_i are similar and step 4 finds all tuples of D, say D', that are similar to e in all attributes of A'. Steps 5-9 compute the similarity score based on the size of the set D'.

Algorithm 2 Computing similarity scores based on the CB similarity function

```
    Input:
        - tuple $e$
        - database $D(a_1, \dots, a_m) = \{t_1, \dots, t_n\}$
    Output: array $Score[n]$, where $Score[i]$ represents the similarity between $e$ and $t_i$
 1: let $A$ be the set of attributes $\{a_1, \dots, a_m\}$
 2: for all $t_i \in D$ do
 3:     let $A' \subseteq A$ be the set of attributes in which $t_i$ is similar to $e$
        //two attributes are regarded as similar if their string similarity is more than a
        //certain threshold $th$
 4:     let $D' \subseteq D$ be the set of tuples that are similar to $e$ in all attributes involved in $A'$
 5:     if $D' \neq \emptyset$ then
 6:         $Score[i] \leftarrow 1/|D'|$
 7:     else
 8:         $Score[i] \leftarrow 0$
 9:     end if
10: end for
```

4.1.2 Cost analysis

Let us analyze the execution time of Algorithm 2. In this algorithm, the first step is to compute for every attribute a_i of e, a set A_i including the tuples of D that are similar to e 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 e 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. $k < logn$, thus the worst case execution time of the algorithm can be written as $O(n^2 \times logn)$. 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.

Algorithm 3 Map function

```
   Input:
      - $\langle e, D \rangle$, where $e$ is an uncertain entity, and $D$ 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 $D$ 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 $\langle key, value \rangle$
```

4.2 Parallel MC

In this section, we propose a parallel version of the MC 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 userŠs map function is a set of $\langle key, value \rangle$ pairs which are collected for MapReduceŠs *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 D, and N as the number of iterations in the MC algorithm, the idea of our MC-MapReduce algorithm is to ask N 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 D, and the required parameters of the MC algorithm (e.g. δ and ϵ) as input and computes N as the number of possible worlds that it should sample to obtain the desired confidence. Then, MC-MapReduce assigns N mappers to do the map function.

Algorithm 3 shows the pseudo code of the map function. This algorithm gets e and 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 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.

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 4 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, say m, and generates the final key-value pair $\langle (t, t'), m \rangle$ as the output of the reduce function.

Algorithm 5 shows the steps which are performed by MC-MapReduce after all mappers and reducers finish their task. This algorithm gets all key-value pairs $\langle (t, t'), m \rangle$ as input. Then, it approximates the probability that each tuple pair (t, t') is the most-probable pair by dividing the

Algorithm 4 Reduce function

```
Input:
    - key $(t, t')$, where $t \in e, t' \in D$
    - value set $V$
$key \leftarrow (t, t')$
$value \leftarrow |V|$
Emit $\langle key, value \rangle$
```

Algorithm 5 Finalize

```
Input: set $S$ of key-value pairs $\langle (t, t'), m \rangle$
Output: most-probable matches
$N \leftarrow \sum_{\langle (t,t'),m \rangle \in S} m$
for all pair $\langle (t, t'), m \rangle \in S$ do
    $P_{(t,t')} \leftarrow m/N$
end for
compute most-probable matches using $P$
```

number of times that it has been the most-similar pair, i.e. m, by N, i.e. the sum of occurrence of all tuple pairs. Using these probabilities, MC-MapReduce approximately computes the most-probable matches.

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, MPM, and MC algorithms in Java. We also implemented the MapReduce version of the MC algorithm in Java using Hadoop framework. We compare CB with Monge-Elkan [23, 24], SoftTFIDF [6] and a version of SoftTFIDF introduced in [19], which we denote as Flexi-SoftTFIDF. Monge-Elkan and SoftTFIDF have been shown to perform better than the other similarity functions [6]. We also consider Flexi-SoftTFIDF. We used the SecondString Java package[3] for implementing SoftTFIDF, Flexi-SoftTFIDF, and Monge-Elkan. The SoftTFIDF and Flexi-SoftTFIDF functions use a similarity function for finding similar tokens inside attributes. For this purpose, we used Jaro-Winkler similarity function [30] 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

[3] http://secondstring.sourceforge.net

other than expanding the possible worlds, i.e. the naive approach. As a result, we consider the naive approach as the only competitor to our approach.

The datasets that we use for comparing the similarity functions are listed in Table 1. The Cora dataset includes bibliographical information about scientific papers in 12 different attributes. The Restaurant dataset includes the *name*, *address*, *city*, and the *type* attributes of restaurants.

We use a number of wordlists[4] for the attribute values of the synthetic database. To generate the database, we use one wordlist, say wordlist L, for each attribute and distribute the words in that wordlist over that attribute using a Gaussian distribution with a mean of $|L|/2$ and a deviation of $|L|/6$, where $|L|$ is the size of the wordlist L.

Table 1: Datasets used in experiments. Source is the place where the dataset originally introduced in, k the number of attributes, n the number of tuples of the dataset, and r the number of searches performed on the dataset

Name	**Source**	**k**	**n**	**r**
Cora	[41]	12	1295	1276
Restaurant	[42]	4	864	224
Synthetic	-	10	30	200

To evaluate the performance of MPM and MC algorithms, we use a probabilistic 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 convert it into a probabilistic database as follows. First, we add an attribute named *confidence* to the database and use Gaussian random numbers with a mean of 0.4 and a deviation of 0.2 for its values. Then, 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; if their confidence values add up to more than 1, we relinquish them and take another set of tuples until we form a valid x-tuple. We repeat this process until we group all tuples in valid x-tuples. We use the default value $d_x = 2$ for the database and also for the uncertain entity unless specified otherwise. We generate the uncertain entity needed for experiments, in the same way that we generated a valid x-tuple.

To emulate a context-free similarity function, we repeatedly pick at random a uniform random number in $[0, 1]$ and attach it as an attribute named A to a tuple. We use the difference between A attributes as the similarity of two tuples. When comparing the MPM algorithm with the naive approach, we use the aforementioned similarity function in both approaches.

When comparing the MC algorithm with the naive approach, we use CB similarity function in both approaches. 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 and real datasets, thus covering all practical cases. Our test method for real datasets is as follows. We repeatedly pick a tuple t from dataset D. If t

[4] ftp://ftp.ox.ac.uk/pub/wordlists

has at least one duplicate tuple in D, except itself, we give the task of finding the most similar tuple in $D - \{t\}$ to t, to the similarity function. If the similarity function returns a 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 use success-rate as one of the performance metrics for comparing the efficiency of different similarity functions.

Our approach for the synthetic dataset is as follows. We repeatedly pick a tuple t from dataset D at random. Then, we convert t to t' by introducing error in a number of attributes of t and give the task of finding the most similar tuple in D to t', to the similarity function. If the similarity function returns t, i.e. the original tuple before introducing error, as the most similar tuple to t', we denote the search as a successful search. To control the process of introducing error into a tuple and the whole test process, we introduce a couple of parameters as follows. The number of times we repeat the test process is denoted as r; the similarity threshold for considering two attributes as matching attributes is denoted as T; and the number of matching attributes, which are considered similar to their erroneous version, is denoted as m. We use the default values $r = 200$, $T = 0.9$, and $m = 2$ unless specified otherwise. To introduce error in a tuple, we pick m attributes of the tuple at random and introduce an amount of error, i.e. less than T, in them to be considered as matching attributes. For other attributes of the tuple, we introduce an amount of error, more than T, in them to be considered as non-matching attributes.

In addition to the success-rate, we also measure the *response time* of different similarity functions. 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 MPM and MC algorithms, we measure their response time and compare it with the naive approach.

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[5] which consists of 20 machines each with Dual core 2.6 GHz CPU and 16 GB memory.

5.2 Results

5.2.1 Performance of MPM and MC algorithms

In this section, we study the performance of MPM and MC algorithms. In our experiments, we set the parameters of the MC algorithm (see Section 3.2.2), as follows. We set N_0 to 500, δ to 0.1, and ϵ to 0.05.

With n increasing up to 2,000,000, Figure 4 compares the response time of the naive approach with that of MPM. This figure shows that the response time of the naive approach increases exponentially with n but that of MPM increases very smoothly.

Using CB as the similarity function, Figure 5 shows the response time of the naive approach and MC. This figure shows that the response time of the naive approach increases dramatically with increasing n while that of MC increases linearly with the number of tuples in database.

We conducted experiments to study the effectiveness of different context-sensitive similarity functions in the MC algorithm. Figure 6 shows the response times of two implementations

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

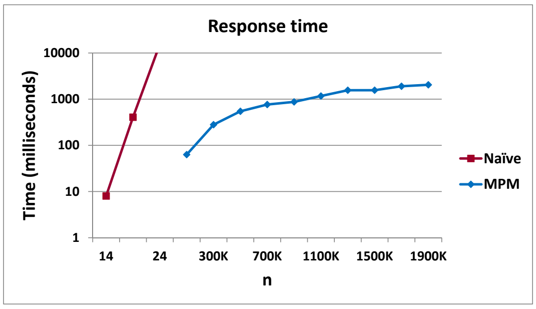

Figure 4: Response times of Naive and MPM

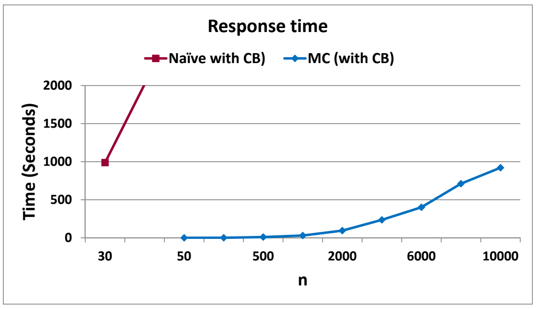

Figure 5: Response times of Naive and MC

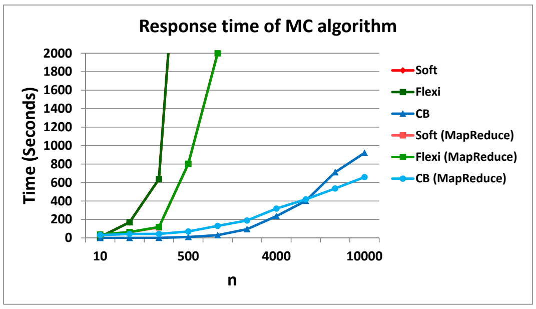

Figure 6: Response times of different implementations of MC

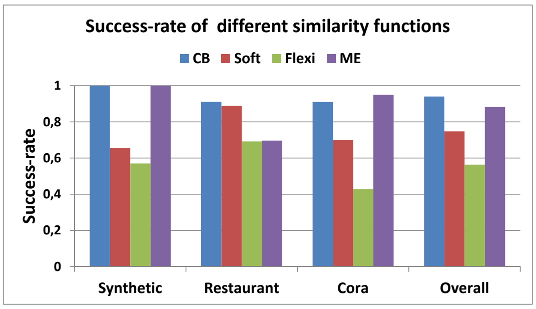

Figure 7: Success-rate of similarity functions over different datasets

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, which are almost the same, increases dramatically with increasing n, while that of CB increases smoothly. Figure 6 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.

5.2.2 Performance of CB

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

Figure 7 compares the success-rate of the CB, SoftTFIDF, Flexi-SoftTFIDF, and Monge-Elkan similarity functions over the three datasets listed in Table 1. Over synthetic dataset, CB and Monge-Elkan achieve the highest possible success-rate. CB outperforms the other similarity functions over Restaurant dataset but the success-rate of SoftTFIDF is only 2% less than that of CB. Monge-Elkan outperforms the other similarity functions over Cora dataset but the success-rate of CB is only 4% less than that of Monge-Elkan. Figure 7 also shows that CB performs best on average.

Figure 8 shows the average setup time and average response time of different context-sensitive similarity functions over different datasets. Overall, CB has zero setup time; the setup

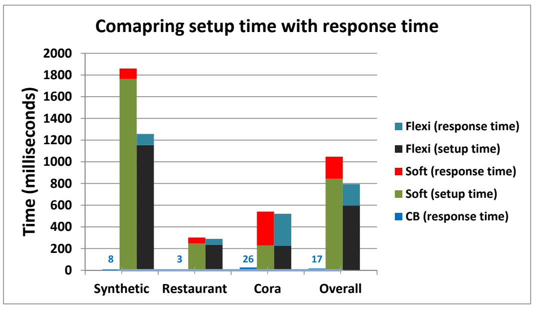

Figure 8: Average setup time and response time of different context-sensitive similarity functions over different datasets

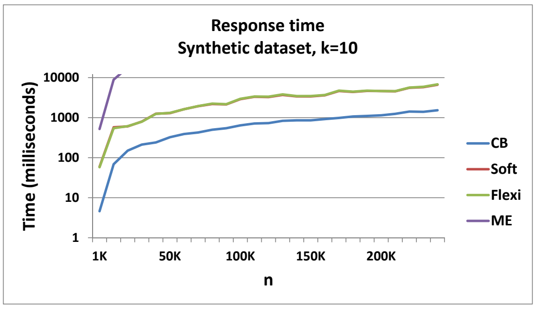

Figure 9: Response time of different similarity functions with increasing *n* up to 240,000

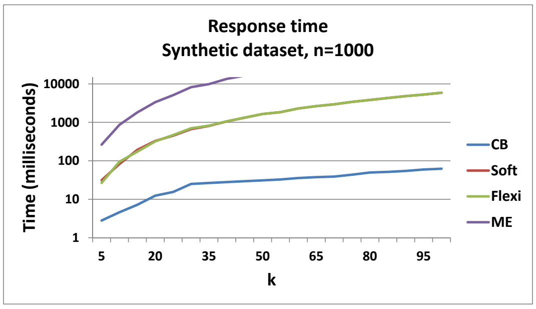

Figure 10: Response time of different similarity functions with increasing *k* up to 100

time of SoftTFIDF is 4 times of its response time; and Flexi-SoftTFIDF's setup time is about 3 times of its response time. 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). To be fair in these experiments, we do not consider the setup time of these similarity functions in the measurements. Figure 9 shows the response time of a search over synthetic dataset with increasing *n* up to 240,000 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 10 shows the response time of a search over synthetic dataset with increasing *k* up to 100 and *n* set to 1000. This figure shows that CB significantly outperforms and far scales better than its competing approaches.

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 similarity function 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 computes the scores of tuples in a community instead of computing them one by one. This greatly improves the response time.

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 [14] for a survey). The problem has been presented under various terms such as *entity resolution*, *duplicate detection*, etc. Recently, there have been some proposals dealing with entity resolution over uncertain data [22, 25]. They are related to our work in the sense that they need to find the similarity between entities for deciding whether they are duplicates or not. However, unlike our approach, they do not combine the confidence values and similarities. For example, the proposal presented in [22] ignores the confidence values when computing the similarity and keeps the original tuples in the database even if they are duplicates. The authors of [25] use the confidence values only for normalizing the similarity of the two x-tuples, i.e. mapping it to [0,1] range. This proposal ignores confidence in favor of similarity, while ours considers both similarity and confidence to compute the probability of the event that two x-tuples match. In fact, the proposals in [22] and [25] ignore the probability distribution over the solution space by completely or partially ignoring the role of the confidence values in defining the outcome of the ERUD. In contrast, we used the well-known possible world semantics to obtain the probability distribution over the solution space and define the outcome of the ERUD.

Among the prior work on uncertain data management, uncertain top-k query processing (e.g., [10, 29, 31, 27]) is the closest to ours. The work in [29] 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 [29] for defining the new semantics of entity resolution over uncertain data. The proposal in [31] 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 [6] and [9] for surveys). There also are a number of proposals for comparing tuples (refer to [13] 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 [12], the weighted similarity is computed by assigning static weights to attributes based on their importance. The proposed technique in [19] modifies the normalization step of the well-known TFIDF similarity function to dynamically adjust the attributes' weights based on their relative importance. [15] 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. 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 pass over database for setup. This makes our similarity function more suitable for the ERUD problem.

7 Conclusion

In this paper, we considered the problem of Entity Resolution for Uncertain Data (ERUD), which is crucial for many applications that process uncertain data. We adapted the possible worlds semantics of uncertain data to define the novel concepts of most-probable match pair and most-probable match entity as the outcomes of ERUD. 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 ERUD. 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] S. Abiteboul, B. Kimelfeld, Y. Sagiv, and P. Senellart. On the expressiveness of probabilistic xml models. *VLDB J.*, 18(5), 2009.

[2] P. Agrawal, O. Benjelloun, A. D. Sarma, C. Hayworth, S. U. Nabar, T. Sugihara, and J. Widom. Trio: A system for data, uncertainty, and lineage. In *Proc. of VLDB*, 2006.

[3] L. Antova, T. Jansen, C. Koch, and D. Olteanu. Fast and simple relational processing of uncertain data. In *Proc. of ICDE*, 2008.

[4] M. J. Atallah and Y. Qi. Computing all skyline probabilities for uncertain data. In *Proc. of PODS*, 2009.

[5] O. Benjelloun, A. D. Sarma, A. Y. Halevy, and J. Widom. Uldbs: Databases with uncertainty and lineage. In *Proc. of VLDB*, 2006.

[6] M. Bilenko, R. J. Mooney, W. W. Cohen, P. D. Ravikumar, and S. E. Fienberg. Adaptive name matching in information integration. *IEEE Intelligent Systems*, 18(5), 2003.

[7] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. Robust and efficient fuzzy match for online data cleaning. In *Proc. of SIGMOD Conference*, 2003.

[8] W. W. Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *Proc. of SIGMOD Conference*, 1998.

[9] W. W. Cohen, P. D. Ravikumar, and S. E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *Proc. of IIWeb*, 2003.

[10] G. Cormode, F. Li, and K. Yi. Semantics of ranking queries for probabilistic data and expected ranks. In *Proc. of ICDE*, 2009.

[11] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *Proc. of VLDB*, 2004.

[12] D. Dey, S. Sarkar, and P. De. Entity matching in heterogeneous databases: a distance-based decision model. In *Proc. of 31st Hawaii International Conference on System Sciences*, volume 7, jan 1998.

[13] C. F. Dorneles, R. Gonçalves, and R. dos Santos Mello. Approximate data instance matching: a survey. *Knowl. Inf. Syst.*, 27(1), 2011.

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

[15] S. Guha, N. Koudas, A. Marathe, and D. Srivastava. Merging the results of approximate match operations. In *Prof. of VLDB*, 2004.

[16] T. S. Jayram, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. Zhu. Avatar information extraction system. *IEEE Data Eng. Bull.*, 29(1), 2006.

[17] C. Jin, K. Yi, L. Chen, J. X. Yu, and X. Lin. Sliding-window top-k queries on uncertain streams. *PVLDB*, 1(1), 2008.

[18] B. Kimelfeld, Y. Kosharovsky, and Y. Sagiv. Query evaluation over probabilistic xml. *VLDB J.*, 18(5), 2009.

[19] N. Koudas, A. Marathe, and D. Srivastava. Flexible string matching against large databases in practice. In *Proc. of VLDB*, 2004.

[20] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8), 1966.

[21] M. Magnani and D. Montesi. Uncertainty in data integration: current approaches and open problems. In *Proc. of MUD*, 2007.

[22] D. Menestrina, O. Benjelloun, and H. Garcia-Molina. Generic entity resolution with data confidences. In *Proc. of CleanDB*, 2006.

[23] A. E. Monge and C. Elkan. The field matching problem: Algorithms and applications. In *Proc. of KDD*, 1996.

[24] A. E. Monge and C. Elkan. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *Proc. of DMKD*, 1997.

[25] F. Panse, M. van Keulen, A. de Keijzer, and N. Ritter. Duplicate detection in probabilistic data. In *Proc. of ICDE Workshops*, 2010.

[26] J. Pei, B. Jiang, X. Lin, and Y. Yuan. Probabilistic skylines on uncertain data. In *Proc. of VLDB*, 2007.

[27] C. Re, N. N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *Proc. of ICDE*, 2007.

[28] A. D. Sarma, O. Benjelloun, A. Y. Halevy, and J. Widom. Working models for uncertain data. In *Proc. of ICDE*, 2006.

[29] M. A. Soliman, I. F. Ilyas, and K. C.-C. Chang. Top-k query processing in uncertain databases. In *Proc. of ICDE*, 2007.

[30] W. E. Winkler and Y. Thibaudeau. 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.

[31] K. Yi, F. Li, G. Kollios, and D. Srivastava. Efficient processing of top-k queries in uncertain databases with x-relations. *TKDE*, 20(12), 2008.

[32] M. L. Yiu, N. Mamoulis, X. Dai, Y. Tao, and M. Vaitis. Efficient evaluation of probabilistic advanced spatial queries on existentially uncertain data. *TKDE*, 21(1), 2009.

[33] S. M. Yuen, Y. Tao, X. Xiao, J. Pei, and D. Zhang. Superseding nearest neighbor search on uncertain spatial databases. *TKDE*, 22(7), 2010.