On entity resolution in probabilistic data

Naser Ayat

On Entity Resolution in Probabilistic Data

Naser Ayat

On Entity Resolution in Probabilistic Data

Naser Ayat

On Entity Resolution in Probabilistic Data

Naser Ayat

On Entity Resolution in Probabilistic Data

Academisch Proefschrift

ter verkrijging van de graad van doctor aan de
Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof.dr. D.C. van den Boom
ten overstaan van een door het college voor
promoties ingestelde commissie, in het openbaar
te verdedigen in de Agnietenkapel
op dinsdag 30 september 2014, te 12.00 uur

door

Seyed Naser Ayat

geboren te Esfahan, Iran

Promotors:	prof. dr. H. Afsarmanesh
	prof. dr. P. Valduriez
Copromotor:	dr. R. Akbarinia
Overige leden:	prof. dr. P.W. Adriaans
	prof. dr. P.M.A. Sloot
	prof. dr. A.P.J.M. Siebes
	prof. dr. M.T. Bubak
	prof. dr. ir. F.C.A Groen

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

SIKS Dissertation Series No. 2014-32
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Printed by: GVO printers & designers.

ISBN: 978 90 6464 812 0

to Aylar, and my parents

Contents

Chapter 1

Introduction

1.1 Uncertain Data

There are many applications that produce uncertain data. Untrusted sources, imprecise measuring instruments, and uncertain methods, are some reasons that cause uncertainty in data. Let us provide some examples of uncertain data which arise in different real-life applications.

- **Uncertainty inherent in data.** There are many data sources that are inherently uncertain, such as scientific measurements, sensor readings, location data from GPS, and RFID data.

- **Data integration.** An important step in automatically integrating a number of structured data sources is to decide which schema elements have the same semantics, i.e. schema matching. However, the result of schema matching is also itself often uncertain. For instance, a schema matching approach outputs *"phone"* and *"phone-no"* as matching, while mentioning that they have a 90% chance to have the same semantics.

- **Information extraction.** The process of extracting information from unstructured data sources, e.g. web, is an uncertain process, meaning that extracted information are typically associated with probability values indicating the likelihood that the extracted information is correct. Consider, for instance, an extractor which decides with 0.95 confidence that Bob lives in Amsterdam.

- **Optical Character Recognition (OCR).** The aim of OCR is to recognize the text within handwritten or typewritten scanned documents. The output of OCR is often uncertain. For instance, an OCR method, which cannot recognize, with certainty, the 'o' character in the word *"Ford"*, may produce the set $\{($*"Ford"* $: 0.8), ($*"F0rd"* $: 0.2)\}$ as output, meaning that

each of the two output words may represent the original scanned word with the associated probability.

Broadly speaking, there are two approaches in dealing with uncertain data: 1) ignoring it, and 2) managing it. While the former approach results in loss of information which is contained in uncertain data, in recent years, we have been witnessing much interest in the latter approach. As shown in the above examples, most real-life applications tend to quantify data uncertainty with probability values, referred to as *probabilistic data*. We also deal with probabilistic data in this thesis.

1.2 Entity Resolution

Entity Resolution[1] (ER) is the process of identifying duplicate tuples, refereing to the tuples that represent the same real-world entity. The ER problem arises in many real-life applications, such as data integration. Consider, for instance, that the large company A acquires another large company B. It is quite likely that A and B share many customers. On the other hand, it is not reasonable to expect a person, who is the customer of both companies, to be represented by identical tuples in the databases of the two companies. For example, consider tuple t_A = ("Thomas Michaelis", "45, Main street") in A's database and tuple t_B = ("T. Michaelis", "45, Main st., 1951") in B's database, both representing the same customer.

In the above example, it is clear for humans that t_A and t_B refer to the same person. However, it is not reasonable to expect the humans to solve the ER problem on large datasets, since every tuple in the dataset should be compared against all other tuples in the same dataset, giving the ER process an $O(n^2)$ complexity. On the other hand, computational power exists on computers to deal easily with the ER problem on large datasets, although they lack the human intelligence, which greatly simplifies the identification of duplicate tuples.

The applications of ER are indeed widespread. For instance, a news aggregation website, such as Google News, automatically gathers news from different sources on the internet. In such websites, identifying the news that refer to the same story is crucial, because otherwise a story is presented to the user over and over again. Another example is comparison shopping websites, such as the Kieskeurig.nl, which aim at presenting the lowest price for any user-specified product. Comparison shopping websites usually aggregate their product items from a number of vendors. Identifying product items that refer to the same product is an inevitable task in such websites.

[1]The ER problem has been referred by various names in the literature, such as *record matching*, *reference matching*, *merge/purge*, *object identification*, *reference reconciliation*, and others.

Due to its diverse applications, the ER problem has a number of different problem definitions. The three main ER problem definitions, denoted by *identity resolution*, *deduplication*, and *record linkage*, are defined as follows.

1.2.1. Definition. *Identity resolution.* Given database D and tuple $e \notin D$, the *identity resolution* problem is to find tuple $t \in D$, where e and t represent the same real-world entity. Typically, it is assumed that D is clean, i.e. does not contain duplicate tuples.

1.2.2. Definition. *Deduplication.* Given database D, the *deduplication* problem is to cluster D's tuples into a number of clusters, where each cluster contains only duplicate tuples that represent the same real-world entity. Once deduplication is done, duplicate tuples in each cluster are merged into a single tuple. In a variation, duplicate tuples are kept in their original form, but the knowledge that which tuples are duplicates of the same entity, is added to the database.

1.2.3. Definition. *Record linkage.* Given D and D' databases, the *record linkage* problem is to find tuple pairs (t, t'), where $t \in D$ and $t' \in D'$ represent the same real-world entity. It is typically assumed that there are no duplicate tuples in the same database.

Each of the above variations of the ER problem happens in a particular setting. As an example setting in which the identity resolution arises, consider aggregating product items from a number of vendors in a comparison shopping website, where each product item is matched against a product catalog to identify which product item represents which product. The deduplication variation mostly arises when integrating not-necessarily-clean databases, which is in contrast to record linkage variation which arises when integrating databases, each of which is clean by itself.

The aim of deduplication is to improve the quality of the database and, as a result, the quality of the queries over it. It is clear that identifying duplicate tuples and considering them in processing the queries significantly improves the quality of aggregate queries. Consider, for instance, the result of a COUNT query over a database with duplicate tuples and its clean version. It is clear that the former yields an erroneous result since each qualifying entity is counted as many times as its duplicate tuples appear in the database, while it should be counted only once.

The query result improvement is not limited to aggregate queries, because non-aggregate queries are also benefited from the aggregated knowledge, resulted from merging the duplicate tuples. For instance, consider the aggregated knowledge that *John McPherson* is a *database researcher* who works for IBM, resulted from identifying and merging duplicate tuples ("John McPherson", "Database researcher", $\perp$) and ("J. McPherson", $\perp$, IBM), where symbol $\perp$ denotes the null value.

1.3 Entity Resolution for Probabilistic Data

The ER problem is important for both deterministic (ordinary) and probabilistic databases, i.e. databases in which there is a probability value associated to the tuples or their attributes. The deterministic case has been widely investigated, so in this thesis we focus on ER in probabilistic databases. There are many applications that need to match a probabilistic entity against a probabilistic database, i.e. identity resolution over probabilistic data. Let us provide an example of such applications from the internet monitoring domain.

1.3.1. Example. *Automatic detection of malicious internet users.* Malicious internet users often use free e-mail service providers, such as Yahoo! Mail and Gmail, to exchange e-mails. The reason is that most of the free e-mail service providers collect minimum information from their users and do not even validate the provided information, which thus allows malicious internet users to hide their real identity. In order to reveal the identity of malicious internet users, an internet monitoring system is used to automatically gather intelligence about internet users and monitor their e-mails. The system can be composed of two main components: 1) user profiling component, and 2) e-mail monitoring component, where the former can be deployed on both internet service providers (ISPs) and mail servers, and the latter only on mail servers. The user profiling component, which is deployed on ISPs, builds for each user an *internet profile*, e.g. using his online public profiles such as Facebook and LinkedIn profiles, weblog posts, tweets, and his internet browsed contents. The user's internet profile may include attributes such as *education*, *job*, *ethnicity*, *height*, *weight*, *fake name*, *hobbies*, etc. The user profiling component uses information extraction tools to automatically extract user's profile from her on internet data. Since the output of information extraction tools is often uncertain, each extracted user's internet profile is thus an uncertain entity, which might be represented using a number of tuples each associated with a probability value indicating its likelihood of truth. Thus, a probabilistic database containing uncertain profiles of ISP's users is maintained at each ISP. An instance of the user profiling component is deployed at each mail server, where an uncertain internet profile is built for each user using his exchanged e-mails on the mail server. The e-mail monitoring component, which is deployed on each mail server, automatically searches each e-mail for certain suspicious keywords, and if the e-mail contains them, matches the uncertain internet profile of the sender of the e-mail against the uncertain profiles of ISPs' users to find the person who *most probably* is the sender of the e-mail, i.e. the suspect.

The above example motivates the need for dealing with the identity resolution problem in probabilistic data. However, identity resolution is not the only variation of the ER problem that arises in probabilistic data. When integrating a

number of probabilistic data sources, it is quite likely that multiple probabilistic tuples represent the same real-world entity, i.e. arising of the need for the ER's deduplication variation in probabilistic data.

1.4 Research Questions

The goal of this thesis is to deal with the ER problem over probabilistic data. In general, we investigate the following:

> *How can we effectively and efficiently deal with the entity resolution problem in probabilistic data (called ERPD)?*

To answer the above general question, we consider different variations of the ER problem over probabilistic data, which results in a number of more specific research questions. We first consider the ER's identity resolution problem variation in probabilistic data.

In identity resolution over *deterministic* data, the aim is to match the *most similar* tuple in the database to the given tuple. However, in matching probabilistic entities, e.g. matching uncertain user profiles in Example 1.3.1, the aim is not clear, since we have to deal with two concepts: *most similar* and *most probable*, at the same time. This leads us to the first research question that we address in this thesis:

1. *How to match a probabilistic entity against a set of probabilistic entities, while considering both their similarity and probability? In other words, what are the semantics of identity resolution problem over probabilistic data?*

We answer this question in Chapter 3 by defining the semantics of the identity resolution problem over probabilistic data, using the *possible worlds* semantics of uncertain data, which treats a probabilistic database as a probability distribution over a set of *deterministic* database instances, each of which is called a *possible world*. The number of possible worlds of a probabilistic database might easily be exponential, which thus makes the computation of the defined semantics impractical. This leads us to our second research question in this thesis:

2. *How can we efficiently deal with the identity resolution problem over probabilistic data?*

This question is also answered in Chapter 3, where it is assumed that the probabilistic data is stored in a centralized database. The relaxation of this assumption leads us to the third research question that we address in this thesis:

3. *How can we efficiently deal with the identity resolution problem over probabilistic data in distributed systems?*

This question is investigated in Chapter 4, where we propose a fully distributed algorithm that efficiently deals with the identity resolution problem over probabilistic data in distributed systems.

Let us now consider the ER's deduplication problem variation in probabilistic data. Similar to *deterministic* data, the aim of deduplication in probabilistic data is to improve the quality of the database. On the other hand, same as in information theory, where the amount of uncertainty in a random variable can be used to represent its quality, the amount of uncertainty in a probabilistic database represents its quality, meaning that the more uncertain the database, the lower its quality. This leads us to the fourth research question that we address in this thesis:

4. *Does deduplication necessarily improve the quality of a probabilistic database? If not, then how can we improve the quality of a probabilistic database through deduplication?*

We answer this question in Chapter 5, where we propose a method for improving the quality of a probabilistic database through deduplication.

In many applications that the ER problem arises, data resides in a number of heterogenous data sources. Consider, for instance, the identity resolution problem in Example 1.3.1, where a more realistic assumption is that mail servers and ISPs are allowed to freely choose a user profiling software. In such a case, it is quite likely that user profiles are represented in heterogenous schemas, which thus, makes the matching of heterogeneous schemas, referred to by *schema matching*, an inevitable step in matching the user profiles. On the other hand, while effective dealing with the schema matching problem is crucial for dealing with the ER problem, it requires human knowledge, which is in contradiction to the fully automated setting of most of the applications in which the ER problem arises.This leads us to the fifth research question that we address in this thesis:

5. *How effectively can we deal with the schema matching problem in a fully automated setting?*

We answer this question in Chapter 6. In order to make our approach generic, we deal with schema matching as one of the main problems that has to be dealt with in a typical data integration system. Thus, in Chapter 6, we aim at building a data integration system in the pay-as-you-go setting in which the system is set up with an acceptable quality in a complete automatic setting and the quality is improved when needed.

1.5 Roadmap

The rest of this thesis is structured as follows.

Chapter 2 provides preliminary definitions and concepts that are used throughout the rest of the thesis. We first present the uncertain data models, and then review the related work on entity resolution.

In Chapter 3, we deal with the identity resolution problem over probabilistic data. We first define the semantics of identity resolution problem in probabilistic data, and we then deal with the problem of efficient computation of the defined semantics, and speeding up the proposed algorithms.

Chapter 4 deals with the identity resolution problem over distributed probabilistic data. We aim at proposing a fully distributed algorithm for computing the semantics of the identity resolution problem, as defined in Chapter 3, in a distributed system. Our primary goal, in the proposed algorithm, is to minimize the bandwidth usage.

In Chapter 5, we deal with deduplication problem over probabilistic data with the aim of improving the quality of probabilistic data. We use the amount of uncertainty in a probabilistic database as a quality metric and aim at minimizing it, which thus maximizes its quality.

Chapter 6 aims at building a data integration system in a fully automated setting. The main problem that is dealt with in this chapter is the schema matching problem, which arises in many applications that need to deal with the entity resolution problem. The possibility of using functional dependencies for matching the schema elements is investigated in this chapter.

Finally, Chapter 7 concludes and gives some research directions for future work.

Chapter 2

Background and Related Work

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

2.1 Uncertain Data Models

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

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

2.1.1 Possible Worlds Semantics

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

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

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

	p-name	**price ($)**
t_1	a	25
t_2	b	$\{(10 : 0.4), (20 : 0.6)\}$
t_3	c	15
t_4	d	$\{(25 : 0.2), (35 : 0.8)\}$
t_5	e	30

(a)

	p-name	**price**
t_1	a	25
t_2	b	10
t_3	c	15
t_4	d	25
t_5	e	30

$w_1 : 0.08$

	p-name	**price**
t_1	a	25
t_2	b	10
t_3	c	15
t_4	d	35
t_5	e	30

$w_2 : 0.32$

	p-name	**price**
t_1	a	25
t_2	b	20
t_3	c	15
t_4	d	25
t_5	e	30

$w_3 : 0.12$

	p-name	**price**
t_1	a	25
t_2	b	20
t_3	c	15
t_4	d	35
t_5	e	30

$w_4 : 0.48$

(b)

SELECT *p-name FROM* $\mathcal{D}$
WHERE *price* < 20

(c)

	p-name	**probability**
t_2	b	$0.08 + 0.32 = 0.4$
t_3	c	$0.08 + 0.32 + 0.12 + 0.48 = 1.0$

(d)

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

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

In Chapter 3, we use the possible worlds semantics to define the semantics of entity resolution in uncertain data.

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

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

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

2.1.2 X-relation Data Model

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

To illustrate, Figure 2.2 shows an example database $\mathcal{D}$ in the x-relation model, and its possible worlds. $\mathcal{D}$ consists of two x-tuples x_1 and x_2, where x_1 consists of two alternatives t_1 and t_2, and x_2 consists of only one alternative t_3. The x-relation's assumptions, i.e. mutual exclusion of alternatives and independence of x-tuples, greatly simplify the computation of possible worlds' probabilities. For instance, the probability of the possible world W_3, denoted by $P(W_3)$, is computed as the joint probability of two independent probabilistic events: among the alternatives of x_1, t_2 occurs; and none of the alternatives of x_2 occur. Thus,

as shown in Figure 2.2(b), $P(W_3)$ is equal to $P(t_2) \times (1 - P(t_3)) = 0.24$. Notice that the probability that none of the alternatives of an x-tuple, say x, occur is equal to $1 - \sum_{t \in x} P(t)$.

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

x-tuple	t	$P(t)$
x_1	t_1	0.6
	t_2	0.3
x_2	t_3	0.2

(a)

w_i	w_i members	$P(w_i)$
w_1	$\varnothing$	$(1 - P(t_1) - P(t_2)) \times (1 - P(t_3)) = 0.08$
w_2	$\{t_1\}$	$P(t_1) \times (1 - P(t_3)) = 0.48$
w_3	$\{t_2\}$	$P(t_2) \times (1 - P(t_3)) = 0.24$
w_4	$\{t_3\}$	$(1 - P(t_1) - P(t_2)) \times P(t_3) = 0.02$
w_5	$\{t_1, t_3\}$	$P(t_1) \times P(t_3) = 0.12$
w_6	$\{t_2, t_3\}$	$P(t_2) \times P(t_3) = 0.06$

(b)

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

2.2 Entity Resolution

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

- **Data preparation:** deals with the heterogeneity in the data coming from different sources.
- **Matching:** finds the duplicate tuples, i.e. the tuples that refer to the same real-world entity.
- **Merging:** merges the duplicate tuples into a merged tuple.

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

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

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

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

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

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

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

In the merging phase, an important challenge is merging duplicate tuples with conflicting attribute values. We distinguish between two types of conflicts: the conflicts that can be resolved by the merge function (i.e. resolvable), and those that cannot be resolved by the merge function (i.e. non-resolvable). Resolvable

conflicts may occur in cases where the duplicate tuples agree on the value of an attribute but they use different naming conventions for representing the attribute, e.g. consider "VLDB" and "very large databases" as two conflicting attribute values. Resolvable conflicts may even occur in cases that the duplicate tuples disagree on the value of an attribute, but the merge function resolves the conflict by combining the conflicting values. For instance, consider two conflicting attribute values 30 and 35 representing the age of a person, and a merge function that resolves this conflict by taking the average of these values.

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

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

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

2.2.1 Matching Methods

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

Overview

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

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

A simple pairwise matching approach is to consider two tuples as matching if their similarity is higher than a certain threshold, where the similarity is measured using one of the metrics for measuring the similarity between tuples, some of which are described in Section 2.2.1. A variation of this technique uses two thresholds μ_1 and μ_2, where $\mu_1 < \mu_2$, to mark two tuples as *matching* if their similarity is higher than μ_2; as *possible-match* if it is between μ_1 and μ_2; and as *non-match*, otherwise. The possible-matches then are manually examined by human experts to determine whether they are matches or not.

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

To evaluate the performance of a matching proposal, we can consider the matching problem as a classification problem and use the *precision*, *recall*, and *F1* metrics, which are frequently used to evaluate the performance of the classification algorithms. To illustrate these metrics, let M denote the set of tuple pairs that are matched together by the matching proposal, and M_{true} is the set of tuple pairs that are true matches. Then, *precision* is the fraction of correctly matched tuple pairs to all pairs in M; *recall* is the fraction of correctly matched tuple pairs to all pairs in M_{true}; and $F1$ is the harmonic mean of *precision* and *recall*, i.e.

$$P = \frac{|M \cap M_{true}|}{|M|};\ R = \frac{|M \cap M_{true}|}{|M_{true}|};\ \text{and}\ F1 = \frac{2.P.R}{P+R}$$

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

Attribute Matching

Typographical variations in string data are the most common source of mismatch between duplicate tuples. As a result, the ER's matching phase mostly relies on string matching techniques to find the duplicate tuples. A number of string matching techniques have been proposed in the literature, each of which is suitable

for particular types of string errors or variations. While errors might occur in non-string data as well, the research on matching non-string data is still in its infancy [61].

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

Character-Based Metrics

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

- Edit distance
- Jaro
- Q-grams

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

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

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

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

The *Needleman-Wunsch* distance [98] is another variation of edit distance which defines different costs for each character replacement, insert, and delete operations. Ristad et al. [113] proposed a method for automatically determining Needleman-Wunsch's costs from a set of equivalent strings written in different variations.

Another variation of edit distance is the *Smith-Waterman* distance [120] in which the cost of edit operations at the beginning and the end of the string are more than that of in the middle of the string. This metric can better match the strings that use different titles for names or use the same title in different locations. Consider, for instance, the three strings "Prof. Andrew S. Tanenbaum", "Andrew S. Tanenbaum, Prof.", and "Mr. Andrew S. Tanenbaum", which are matched using the Smith-Waterman metric within a short distance.

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

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

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

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

$$Jaro\text{-}Winkler(s_1, s_2) = Jaro(s_1, s_2) + \frac{1}{10} max\left(LCP, 4\right).(1 - Jaro(s_1, s_2)),$$

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

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

Q-grams are used in a variety of ways for measuring the similarity between strings. For instance, *dice coefficient* metric [34] uses the number of shared q-grams between two strings and the total number of their q-grams for defining the

similarity value. More precisely, the dice coefficient of two strings s_1 and s_2 is defines as:

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

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

Token-Based Metrics

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

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

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

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

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

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

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

them, as the similarity of the strings that the vectors represent. More precisely, the TF-IDF similarity between strings s and s' is defined as:

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

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

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

$$\text{SoftTF-IDF}(s, s') = \frac{\sum\limits_{w \in similar(s,s')} v(w, s).v(w, s').sim(w)}{\sqrt{\sum\limits_{w \in similar(s,s')} v(w, s)^2}.\sqrt{\sum\limits_{w \in similar(s,s')} v(w, s')^2}}$$

Setting μ to one in the above equation results in the same similarity values as the TF-IDF metric.

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

Metrics for Non-String Data

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

Effectiveness of Different Metrics

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

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

Tuple Matching

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

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

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

Distance-Based Approaches

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

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

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

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

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

An important problem of distance-based pairwise approaches is choosing the appropriate distance threshold. Chaudhuri et al. [38] proposed a method to deal

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

Machine Learning-Based Approaches

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

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

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

$$class(\rho) = \arg\max_{c \in \{M,N\}} P(c) \prod P(v_i \mid c)$$

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

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

The basic idea in active learning is that the learning algorithm actively participates in the task of choosing the subset of unlabeled data whose labeling and inclusion in the training set most likely improves the accuracy of the learning algorithm.

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

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

Rule-Based Approaches

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

$$sim(t_1.name, t_2.name) \geqslant 0.9 \wedge t_1.byear = t_2.byear \implies Match(t_1, t_2),$$

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

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

Usability of Matching Methods in ERPD

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

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

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

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

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

2.2.2 Scalability Issues

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

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

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

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

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

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

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

Achieving scalability in dealing with the ERPD problem is more challenging since we have to deal with an exponential number of possible worlds of probabilistic data. Our proposed approaches for dealing with the ERPD problem however are scalable, as shown through computational complexity analysis, and experi-

mentation. Moreover, we believe that our CB similarity function, presented in Chapter 3, can act as a cheap similarity metric in the blocking methods that, as explained above, aim at improving the efficiency of ER in deterministic data. Elaborating on this idea however remains as a possible direction for future research (see Section 7.2).

2.2.3 Entity Resolution for Probabilistic Data

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

Matching X-tuples

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

$$Sim_{exp}(x, x') = \sum_{t \in x} \sum_{t' \in x'} \frac{p(t)}{p(x)} . \frac{p(t')}{p(x')} . Sim(t, t'),$$

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

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

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

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

Merging X-tuples

[102] is a proposal by Panse et al. that addresses the merging of x-tuples in an extended version of the x-relation model, where each x-tuple is associated with a probability value indicating its membership degree to the database. In order to merge two x-tuples, they assign a weight to each x-tuple, where sum of the weights is equal to one; multiply the probability of each tuple by the weight of the x-tuple to which it belongs; unify the tuples of the two x-tuples into the merged x-tuple; and combine the identical tuples by adding their probabilities.

x-tuple	t	name	age	phone-no	p(t)	p(x)
x	t_1	John	35	5256662	0.25	0.8
	t_2	John	30	5256662	0.45	
	t_3	John	$\perp$	7895226	0.3	

(a)

x-tuple	t	name	age	phone-no	p(t)	$p(x')$
x'	t_1	John	35	5256662	0.4	0.5
	t_2	John	30	5256662	0.35	
	t_4	John	32	$\perp$	0.25	

(b)

x-tuple	t	name	age	phone-no	p(t)
$x \odot x'$	t_1	John	35	5256662	$0.25 \times 0.8 + 0.4 \times 0.2 = 0.28$
	t_2	John	30	5256662	$0.45 \times 0.8 + 0.35 \times 0.2 = 0.43$
	t_3	John	$\perp$	7895226	$0.3 \times 0.8 = 0.24$
	t_4	John	32	$\perp$	$0.25 \times 0.2 = 0.05$

(c)

Figure 2.3: Panse et al.'s approach for merging x-tuples

As an example, consider merging two x-tuples x and x' shown in Figures 2.3(a) and 2.3(b), respectively. Assigning weights 0.6 and 0.4 respectively to x and x', the resulted merged x-tuple, denoted by $x \odot x'$, is shown in Figure 2.3(c). The membership probability of $x \odot x'$ is not shown in the Figure since it depends on the interpretation of relationship between source relations, to which x and x' belong, and the destination relation of the $x \odot x'$.

The important drawback of this proposal is that it does not merge tuples that are mergeable[1] but not identical (i.e. tuples with null values), e.g. tuples t_3 and t_4 in Figure 2.3. Thus, the whole data that the database has about an entity, is not aggregated in one tuple, which thus adversely affects the quality of the query results over the database. Our merge function, proposed in Chapter 5, does not suffer from this shortcoming.

Deduplication in Probabilistic Data

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

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

A generic *merge* function is then used to merge the matched tuples into a tuple.

Koosh is built on the assumption that the confidence of match function in matching two tuples affects the probability of the resulted merged tuple. For instance, consider three tuples t_1, t_2, and t_3 all having equal probabilities, and suppose that the match function is 80% sure that t_1 and t_2 are a match, and 40% sure that t_2 and t_3 are a match. Then, the tuple resulted from merging t_1 and t_2 gets higher probability than that of merging t_2 and t_3.

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

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

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

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

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

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

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

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

2.2.4 Conclusion

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

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

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

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

Chapter 3

Entity Resolution for Probabilistic Data[1]

3.1 Introduction

As discussed in Chapter 1, the entity resolution (ER) problem is defined differently in the literature. In this chapter, we focus on the *identity resolution* definition of the ER problem (see Definition 1.2.1).

The ER problem arises in many applications that have to deal with probabilistic data. Before proceeding, let us first motivate the need for ER in probabilistic data (which we call ERPD), with an example from anti-criminal security domain.

3.1.1. Example. *Suspect detection in anti-crime police database.* The anti-crime 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 informants. Nevertheless, some of the gathered data are not certain, for some reasons, Such as the fact that the police cannot completely trust informants and/or witnesses. To represent this uncertainty, probability values can be attached to the data to show their likelihood of truth, according to the confidence in the sources. These probabilistic data are used to find possible suspects, and can also 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 that consists 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 answer to the following query: *find in the uncertain Suspects 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

[1] The material of this chapter has been partially published in [17] and [21].

each one using an uncertain entity and repeat the process for each entity as far as needed.

Existing ER approaches are not appropriate for answering the above query since they match tuples only based on their similarity and completely ignore their probabilities.

Inspired by the literature on uncertain data management, in this chapter we adopt the well-known possible worlds semantics of uncertain data for defining the semantics of the ERPD problem and propose efficient algorithms for computing it. Developing an efficient solution for the ERPD problem however 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, we may find different similarity values between the entity and the tuples of the database. Third, in the case of *context-sensitive* similarity functions (see the definition in Section 3.2.2), the similarity of two entities may be different in different possible worlds. A naïve solution for ERPD involves enumerating all possible worlds of the uncertain entity and the database. However, this solution is exponential to the number of tuples of the database.

In this chapter, 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 high response time of existing context-sensitive similarity 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 have 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 chapter is organized as follows. In Section 3.2, we present our data model, and define the problem we address. In Section 3.3, we propose our

solution to the ERPD problem for context-free similarity functions. In Section 3.4, we propose our solution for dealing with the ERPD problem when the similarity function is context-sensitive. In Section 3.5, we report the performance evaluation of our techniques over synthesis and real data sets. Section 3.6 discusses analysis against related work, and Section 3.7 concludes.

3.2 Problem Definition

In this section, we first give our assumptions regarding the data model, and then define the problem that we have addressed.

3.2.1 Data Model

For representing an uncertain database, we use the x-relation probabilistic data model [10] in which each uncertain entity is represented with an x-tuple (see the definition in Section 2.1.2).

We denote an uncertain database by $\mathcal{D}$, the set of its possible worlds by $PW(\mathcal{D})$, and the set of all tuples in $\mathcal{D}$ by D. We also define the set of possible worlds for 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.

3.2.2 Context-Free and Context-Sensitive Similarity Functions

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 [139], Monge-Elkan [96, 97] and Levenshtein [85] 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. TF-IDF [49] and Ranked-List-Merging [71] are examples of context-sensitive similarity functions.

3.2.3 Problem Statement

While the ER problem is semantically clear in *deterministic* data, its semantics is not clear in probabilistic data since we have to take similarity and probability into consideration. In this section, we use the possible worlds semantics for defining the semantics of the ERPD problem.

The interaction between the concepts of *most similar* and *most probable* makes different definitions possible. The followings are two of the main definitions:

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

In our approach, we deal with the second definition since it takes into account all possible worlds for finding the most similar tuple to the given entity.

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 in which t is the most similar tuple to e. We denote the tuple with maximum P_{msp} as the *most-probable matching tuple* for entity e.

In the case where the entity e has multiple alternatives, the most-probable matching tuple needs to be extended by the alternative of e with which it is the most-probable matching tuple, say pair $(t, t'), t \in e, t' \in D$. We refer to this pair as *most-probable matching pair* and we use the first element of the pair for tuple in entity e and the second element for a tuple in D. In addition to the most-probable matching pair concept, uncertain data enables us to define another new concept, which does not make sense in deterministic data: *most-probable matching entity*. The most-probable matching 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 matching pair and the most-probable matching entity as *most-probable matches*. Let us first intuitively explain these concepts using an example.

3.2.1. Example. Consider the entity e and the *Suspects* database in the Figures 3.1(a) and 3.1(b), respectively. Let *Sim* be a similarity function that ranks all possible tuple pairs of e and *Suspects* as they appear in Figure 3.1(c). Figure 3.1(d) 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 3.1(e). 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 matching pair* since it has the maximum P_{msp} among all other pairs. Figure 3.1(f) shows the result of computing P_{msp} for all

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

(a)

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

(b)

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

(c)

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	$\{(t_{e,2}, t_{1,2}), (t_{e,2}, t_{2,1})\}$	$(t_{e,2}, t_{2,1})$	$0.5 \times 0.16 = 0.08$

(d)

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

(e)

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

(f)

Figure 3.1: a) Uncertain entity e, b) Uncertain database *Suspects*, c) Tuple pairs ranked based on their similarity, d) Possible worlds set of e and *Suspects*, MSP in each world, e) All pairs and their probability to be MSP (P_{msp}), f) Entities and their P_{msp}

entities in the *Suspects* database. This figure shows that e_1 is the *most-probable matching 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.

3.2.2. DEFINITION. *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 Sim be a similarity function for computing the similarity between tuples. Let the most similar pair in 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 Sim. Let ρ be a tuple pair in $e \times D$. Let $P_{msp}(\rho)$ be the aggregated probability indicating 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 matching pair** of e and $\mathcal{D}$, MPMP(e, $\mathcal{D}$), as:

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

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

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

MPMP(e, $\mathcal{D}$) and MPME(e, $\mathcal{D}$) together are called most-probable matches.

Given uncertain entity e, uncertain database $\mathcal{D}$, and similarity function Sim, our goal is to efficiently find the most-probable matches. In Sections 3.3 and 3.4, we present our approach for finding the most-probable matches, respectively, for the context-free and context-sensitive similarity functions.

3.3 Context-Free Entity Resolution

A straightforward solution for entity resolution over probabilistic data is to enumerate all possible worlds of the given entity and the database, i.e. $PW(e, \mathcal{D})$, cumulate for each match the probability of the possible worlds where the match is the most similar, and finally return the match that has the highest probability to be the most similar. However, this approach does not scale due to the exponential number of possible worlds.

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

3.3.1 CFA Algorithm

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. $S = e \times D$, and $\rho = (t, t_i)$ be a tuple pair in S. Since alternative tuples of e are mutually exclusive, there is no possible

world in $PW(e, \mathcal{D})$ containing the 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 t be an alternative tuple of the entity e, and $L = \{t_1, \ldots, t_n\}$ be the list of D tuples sorted based on their similarity with t 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 the tuples t_1 to t_i, only the t_i occurs. Considering x-tuple correlations in calculating the probability of the latter event, we can calculate $P_{msp}(t, t_i)$ as:

$$P_{msp}(t, t_i) = P(t) \times P(t_i) \times \prod_{x \in X_i} (1 - P(x)) \tag{3.1}$$

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, if any, is omitted from it. Using Equation (3.1), our CFA algorithm computes P_{msp} of all tuple pairs in set S and then uses P_{msp}s to compute the most-probable matches.

Algorithm 1 describes the details of our approach 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 Equation (3.1), 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 the $L[i]$ 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 (3.1), 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 matching pair by finding the record in set R with the maximum value for the P_{msp} field. The most-probable matching 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 6-10 takes $O(i + N)$ time, where N

Algorithm 1 : CFA algorithm for context-free entity resolution

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

Output: Most-probable matches

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

is the number of x-tuples in $\mathcal{D}$. Thus, steps 5-11 take $\sum_{i=1}^{n} O(i+N)$ time, which is $O(n^2)$ since $N \leqslant n$. Therefore, steps 4-11 are done in $O(n^2)$. Since these steps are repeated m times (i.e. the number of alternatives of e), steps 3-12 take $O(m \times n^2)$ time. Step 13 takes $O(m \times n)$ time, which is dominated by $O(m \times n^2)$. Thus, the total execution cost of the algorithm is $O(m \times n^2)$.

3.3.2 Improving CFA Algorithm

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

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

As we will show in Section 3.5, the number of visited pairs in CFA-MPMP and CFA-MPME algorithms is significantly less than that of the CFA algorithm. As a result, to further speedup these algorithms, we do not sort pairs based on their similarity scores in Step 4 of Algorithm 1, but instead we implement an iterator interface to incrementally provide next pair with the highest score, when it is needed by the algorithm.

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 an upper bound on the P_{msp} values of the pairs that come after $L[i]$ in the 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 the list L. The following lemma computes such upper bound.

3.3.1. Lemma. *Let t be an alternative of the uncertain entity e. Let L with size n be the list of D tuples sorted based on their similarity with t in descending order. Then:*

$$\forall i \in [1..n], P_{msp}(t, L[i]) \leqslant P(t) - \sum_{j \in [1..i-1]} P_{msp}(t, L[j]) \tag{3.2}$$

Proof. We first show that:

$$\sum_{j \in [1..n]} P_{msp}(t, L[j]) = P(t). \tag{3.3}$$

Let W_t be the subset of possible worlds that contain tuple pairs in the from of $(t, L[j]), j \in [1..n]$, i.e. $W_t = \{w \mid w = \{t\} \times v, v \in PW(\mathcal{D})\}$. We have:

$$\begin{aligned} LHS(3.3) &= \sum_{w \in W_t} P(w) = \sum_{v \in PW(\mathcal{D})} P(t) \times P(v) \\ &= P(t) \times \sum_{v \in PW(\mathcal{D})} P(v) = P(t) = RHS(3.3). \end{aligned}$$

Thus, $P(t)$ is an upper bound on $P_{msp}(t, L[i])$. However, to obtain a tighter upper bound on $P_{msp}(t, L[i])$, we can deduct the probability of possible worlds in which a tuple pair other than $(t, L[i])$ is the most similar pair. This means that $P(t) - P_{msp}(t, L[j]), j \neq i$, is an upper bound on $P_{msp}(t, L[i])$, and so is RHS(3.2). □

3.3.2. Corollary. *Let t be an alternative of the uncertain entity e. Let L with size n be the list of D tuples sorted based on their similarity with t in descending order. Then:*

$$\forall i \in [1..n], \forall j \in [i..n], P_{msp}(t, L[j]) \leqslant P(t) - \sum_{k \in [1..i-1]} P_{msp}(t, L[k])$$

Corollary 3.3.2 implies that the computed upper bound on $P_{msp}(t, L[i])$ in Lemma 3.3.1 is also an upper bound on the P_{msp} value of the pairs that come after $L[i]$ in the list L. Using Corollary 3.3.2, the CFA-MPMP algorithm can stop early in visiting the pairs of the list L. Let us illustrate the CFA-MPMP algorithm using an example.

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	$\mathrm{P}_{\mathrm{msp}}(t_{e,1}, t)$	upper bound
$t_{2,2}$	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

Figure 3.2: a) An uncertain entity e, b) A database $\mathcal{D}$, c) List L, $\mathrm{P}_{\mathrm{msp}}$, and upper bound values

3.3.3. Example. Consider uncertain entity e and uncertain database $\mathcal{D}$ shown in Figures 3.2(a) and 3.2(b), respectively. Let *Sim* be a context-free similarity function. Let L, shown in Figure 3.2(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*.

After visiting the first tuple in L, the value of $P_{msp}(t_{e,1}, t_{2,2})$ is computed as 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 upper bound is equal to $P(t_{e,1}) - P_{msp}(t_{e,1}, t_{2,2})$ which is equal to 0.072. This is an upper bound on the P_{msp} values that we obtain if we continue processing list L, and since the computed upper bound is greater than the yet computed maximum P_{msp} value, i.e. 0.08, we continue processing the list.

After visiting the second tuple in L, the value of $P_{msp}(t_{e,1}, t_{4,1})$ is computed as 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 upper bound is equal to $P(t_{e,1}) - P_{msp}(t_{e,1}, t_{2,2}) - P_{msp}(t_{e,1}, t_{4,1})$ which is equal to 0.216. At this point, we stop processing the list since the computed upper bound is already 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 hence, is $MPMP(e, \mathcal{D})$. Notice that we only process two pairs out of the whole 16 pairs for computing $MPMP(e, \mathcal{D})$. Thus, we gain a very good improvement, compared to the basic version of the CFA algorithm.

CFA-MPME Algorithm

We know that the sum of the P_{msp} values of all pairs is a constant. Therefore, we use this fact to stop early in the CFA algorithm when the probability of the so far computed most-probable matching entity, say entity $e_{max} \in \mathcal{D}$, is high enough

that none of the other entities in the database can obtain such probability. Such condition holds when even if all of the uncomputed P_{msp} values goes to each of the competitors of e_{max}, the probability of that competitor is still less than that of e_{max}. The following lemma defines the condition for stopping early in computing MPME using the CFA algorithm.

3.3.4. LEMMA. *Let V be the set of pairs that we have computed their P_{msp} values 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.*

$$P_{agg}(e_i) = \sum_{(t,t')\in V, t'\in e_i} P_{msp}(t,t').$$

Let e_{max} be the entity with maximum P_{agg} value. Then:

$$\forall e_i \in \mathcal{D}, e_i \neq e_{max}, P_{agg}(e_i) + \left(\sum_{t\in e} P(t) - \sum_{\rho\in V} P_{msp}(\rho)\right) < P_{agg}(e_{max}) \Rightarrow e_{max} = MPME(e,\mathcal{D}). \tag{3.4}$$

Proof. We know that sum of the P_{msp} of all pairs is equal to sum of the probabilities of e's alternatives, i.e. $\sum_{\rho\in e\times D} P_{msp}(\rho) = \sum_{t\in e} P(t)$.
Thus, the term $\left(\sum_{t\in e} P(t) - \sum_{\rho\in V} P_{msp}(\rho)\right)$ in LHS(3.4) is equal to the sum of the uncomputed P_{msp} values, and LHS(3.4) means that even by adding this value to each of the entities in the database other than e_{max}, their probability to be MPME is still less than that of e_{max}. As a result, based on Definition 3.2.2, it is clear that if the condition in LHS(3.4) holds, e_{max} is equal to $MPME(e,\mathcal{D})$. □

We check the stop condition after computing P_{msp} in Step 9 of Algorithm 1, and the algorithm ends if the condition holds.

As an example consider uncertain entity e and uncertain database $\mathcal{D}$ shown in Figures 3.2(a) and 3.2(b) respectively. The sum of the probabilities of e's alternatives is equal to 0.9. After visiting the first two pairs of list L, shown in Figure 3.2(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 non-computed P_{msp}s is equal to $0.9 - 0.584 = 0.316$. The entity e_4 is MPME since even if all non-computed P_{msp}s are from an entity other than e_4, its P_{agg} is still less than that of e_4.

3.4 Context-sensitive Entity Resolution

In this section, we focus on context-sensitive similarity functions 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.

3.4.1 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 TF-IDF, 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 to the chosen entity alternative. 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 $P\{-z \leqslant N(0,1) \leqslant z\} = 1-\delta$, $N(0,1)$ 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.

3.4.2 Parallel MC

In this subsection, we propose a parallel version of our MC algorithm, denoted as MC-MapReduce, which we have developed using the MapReduce framework. Let us briefly introduce MapReduce that is a framework for parallel processing of large datasets in two phases : *map* and *reduce*. 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 e, $\mathcal{D}$, 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. 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.

The MapReduce framework receives all generated pairs and sends all pairs with the same key to one reducer. Algorithm 3 shows the pseudo code of the reduce function. This algorithm gets (t, t') as key and the set of values V, which contains a set of 1 values, 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.

3.4.3 CB Similarity Function

Most of the context-sensitive similarity functions in the literature need to compute some statistical features of the data, e.g. the rarity of words, to setup the similarity function. Such similarity functions mostly work at the word or q-gram level, meaning that they need to perform a lot of rather costly string comparison operations during the setup, which result in the low efficiency of these functions. In many applications, spending a significant amount of 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 which is used only once, i.e. for the selected possible world.

To improve the efficiency of the MC algorithm, we propose a new context-sensitive similarity function, called *CB* (Community Based), that significantly reduces the number of string comparison operations by working at the attribute level rather than at the word or q-gram level. An important feature of CB, in contrast to most other matching methods that work at the attribute level, is that it does not need the specification of weights for representing the relative importance of individual attributes. As we will show in Section 3.5, our similarity function significantly outperforms the baseline context-sensitive similarity

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

functions in terms of response time, while offering good performance according to success-rate and F1 metrics.

In CB, we give more importance to more discriminative attributes of tuples by introducing a novel concept called *community*. A community C of a relation D is defined as the subset of all 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. Based on this definition, each attribute partitions tuples into a number of communities, and each tuple belongs to a number of communities based on the values of its attributes. For illustration, consider Figure 3.3(b) which shows a 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'. Notice that a

	Gender	City	Age-range
t	F	A	18-40

(a)

t_i	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)

t_i	$C_{min}(t, t_i)$	Score(t, t_i)
t_1	{t_1,t_3,t_6,t_7,t_{10}}	$1 - (\log 5/\log 10) = 0.3$
t_2	$\varnothing$	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)

Figure 3.3: a) An example tuple t, b) An example relation D, c) the similarity of D tuples to t

tuple that is not involved in a relation can be considered as a member of some communities of the relation. Take as example the tuple t in Figure 3.3(a), which can be considered as a member of the above mentioned communities C_1 and C_2.

In CB, the similarity of two tuples t and t' depends on the size of the smallest community, say community $C_{min}(t, t')$, to which they belong. For instance, considering Figures 3.3(a) and 3.3(b), $C_{min}(t, t_8) = \{t_6, t_7, t_8\}$ and $C_{min}(t, t_1) = \{t_1, t_3, t_6, t_7, t_{10}\}$. Notice that C_{min} of two tuples is their most discriminative community. CB defines the similarity between tuples based on C_{min} as follows.

3.4.1. Definition. *CB similarity function.* Let t be a tuple, D a relation, and $t_i \in D$ a tuple in D. Let $C_{min}(t, t_i)$ be the smallest community to which t and t_i belong. The similarity score of t to t_i, denoted as $score(t, t_i)$, is defined as

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

According to CB, the smaller the size of the smallest community to which the two tuples belong, the more similar are the two tuples, and if they do not belong to any common community, they are not similar at all.

An important property of CB is that the similarity value of two tuples drops as the number of tuples similar to them grows. In other words, a match between two tuples on a set of attributes is more significant if there is less number of other tuples similar to them on those attributes.

Let us now deal with computing the smallest community of two tuples. A naïve way for computing C_{min} of two tuples is to enumerate all communities to which the two tuples belong, and then return their smallest common community. However, this approach does not scale well, because the number of communities to which a tuple belongs is exponential to the number of attributes of the tuple.

Using the following lemma, we propose an efficient method for computing C_{min} of two tuples.

3.4.2. Lemma. *Let t be a tuple, D a relation, and $t_i \in D$ a tuple in D. Suppose f is a similarity function that computes the similarity between two attribute values. Let S_i be the set of attributes in which t is similar to t_i according to the similarity function f, and $D_i \subseteq D$ be the set of tuples that are similar to t in all attributes involved in S_i. Then, $C_{min}(t, t_i) = D_i$.*

Proof. Let $D_a \subseteq D$ be the set of tuples that are similar to t in the attribute $a \in S_i$. According to the definition of D_i we have $D_i = \bigcap_{a \in S_i} D_a$. We show that

$$C_{min}(t, t_i) = \bigcap_{a \in S_i} D_a. \tag{3.5}$$

The D_a communities, where $a \in S_i$, are the only communities to which both tuples t and t_i belong. Thus, the communities which are based on the attributes that are not in set S_i, cannot contribute in RHS(3.5). We prove Equation (3.5) based on induction on the number of attributes in set S_i, i.e. $|S_i|$. The inductive basis is for $|S_i| = 1$, when it is easy to observe the correctness of (3.5). By the inductive hypothesis, we have

$$C_{min}(t, t_i) = \bigcap_{a \in S_i} D_a, \tag{3.6}$$

where $|S_i| = m - 1$. Let a_m be the attribute which is added to set S_i, and C'_{min} be $RHS(3.6) \bigcap D_{a_m}$. It is clear that both tuples t and t_i belong to $RHS(3.6)$ and D_{a_m}, thus both of them belong to C'_{min}. In addition, $RHS(3.6)$ is the smallest common community of t and t_i on attributes $|S_i| = m-1$. Thus, $RHS(3.6) \bigcap D_{a_m}$ is their smallest common community when we add a_m to S_i. Thus, $C'_{min} = C_{min}(t, t_i)$, where $|S_i| = m$. □

According to Lemma 3.4.2, for computing $C_{min}(t, t_i)$ it is sufficient to find the set of attributes in which t and t_i are similar, and then return the set of D's tuples that are similar to t in those attributes. As an example, Figure 3.3(c) shows C_{min} and similarity of each tuple of relation D to tuple t. 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.

Algorithm 5 CB similarity function

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

Output: array $Score[n]$, where $Score[i]$ represents $Score(t, t_i)$

1: define hash table H and $H \leftarrow \varnothing$
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 according to f
// let $D_i \subseteq D$ be the set of tuples that are similar to t in all attributes involved in S_i
// according to f
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**

CB Algorithm

Algorithm 5 shows the pseudo code of the CB similarity function. It takes as input tuple t, a set of tuples D, and similarity function f, which is used for matching the attribute values. 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 3.3(c), once we

compute the set of tuples that are similar to tuple t in attribute set $\{age\text{-}range\}$, we use it for computing both $score(t, t_1)$ and $score(t, t_{10})$. This greatly reduces the execution cost of the algorithm.

In order to increase the efficiency of the algorithm, we compute a binary matrix, say matrix M, where $M[i][j]$ stores the result of matching the value of jth attribute of t with that of t_i. This avoids multiple computation of the matching result of two attribute values.

Cost analysis

Let us analyze the execution time of Algorithm 5. Suppose n be the number of tuples involved in relation D, and k be the number of attributes that are common in the schemas of t and D. Let S_D be the set of attributes in D's schema. For each tuple $t_i \in D$, the algorithm computes the set of attributes $S_i \subseteq S_D$ in which t_i is similar to t. Then, it finds the set of tuples $D_i \subseteq D$ that are similar to t in all attributes involved in S_i if set D_i has not already been computed. In the worst case, these steps take $O(n \times k)$. These steps are repeated for computing the similarity score of each tuple $t_i \in D$. Thus, in total the CB algorithm is executed in $O(n \times k \times l)$, where l is the number of different S_i sets. Since l is bounded by n, in the worst case, the CB algorithm is executed in $O(n^2 \times k)$. However, our experiments show that l is far less than n. Thus, the average execution time of this algorithm is much better than its worst case performance.

3.5 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 CFA and MC algorithms. Afterwards, we compare the performance of our similarity function with competing approaches. Finally, we summarize the performance results.

3.5.1 Experimental setup

We have implemented our algorithms CFA, CFA-MPME, CFA-MPMP, MC, and our CB similarity function in Java. We implemented the MapReduce version of the MC algorithm using Hadoop framework. We compare CB with Jaccard, Levenshtein, SoftTF-IDF and the soft version of FlexiTF-IDF, which we denote as Flexi (refer to Section 2.2.1 for definitions). We used the SecondString Java package[2] for implementing these similarity functions. The SoftTF-IDF and Flexi functions use a similarity function for finding similar tokens inside attributes.

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

Also, CB uses a similarity function for finding similar attributes. For this purpose, we used Jaro-Winkler similarity function for all of these methods and we used the same similarity threshold for all of them.

Table 3.1: Datasets used in experiments. Source is the place in which the dataset was originally introduced, k is the number of used attributes, and n is the number of tuples of the dataset

Name	**Source**	**k**	**n**
Cora	[92]	3	1,295
Restaurant	[124]	4	864
Census	[76]	6	3,000
Synthetic	-	5..1,500	10..2,000,000

The benchmark datasets that we use for our experiments are listed in Table 3.1. The Cora, Restaurant, and Census datasets have been frequently used in the literature to evaluate similarity functions, e.g. [92, 124, 76, 31, 58, 30]. 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 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 database which we generate ourselves.

We use a number of wordlists[3] 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.

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 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 $[1, d_x]$, 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

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

choose one of the x-tuple's alternatives as the seed tuple, say t, and 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[4] 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 [139] to evaluate the CFA algorithms. When comparing the MC algorithm with the naïve approach, we use CB similarity function in the naïve approach and the MC algorithm. Moreover, to study the performance of different context-sensitive similarity functions in the MC algorithm, we use SoftTF-IDF, Flexi, and CB similarity functions. To the best of our knowledge, in the literature there is no approach for computing most-probable matches other than expanding the possible worlds, i.e. the naïve approach. Thus, in addition to our algorithms, we report the performance results of the naïve approach.

Method and performance metrics. We use success-rate and F-measure (also called F1 score) 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.

To evaluate the scalability of different similarity functions, we measure their *response time*, which is the time that it takes to find the tuple most similar to a

[4]Indeed, we use a continuous Gaussian 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 $P(x \in r_i)$.

given tuple.

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 improved versions (i.e. CFA-MPME and CFA-MPME), we measure the number of tuple pairs which are visited by these algorithms.

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.

3.5.2 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 3.4).

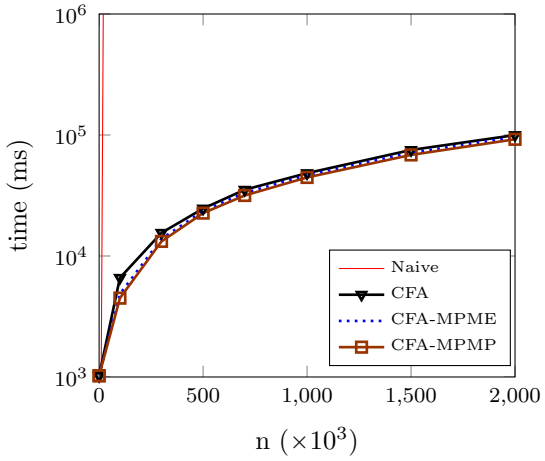


Figure 3.4: Response times of Naïve and CFA algorithms

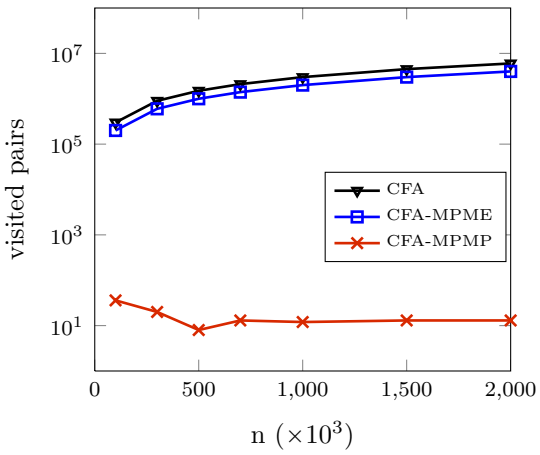


Figure 3.5: The number of visited pairs vs. n in CFA algorithms

With n increasing up to 2,000,000, Figure 3.4 compares the response time of the naïve approach with that of CFA, CFA-MPME and CFA-MPMP. 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 noticeable.In order to better compare the performance gain of CFA-MPME and CFA-MPMP over CFA, Figure 3.5 shows the number of visited tuple pairs in these

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

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 3.6 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 SoftTF-IDF and Flexi. The response times of SoftTF-IDF and Flexi increases dramatically with increasing n, while that of CB increases smoothly. Figure 3.6 also shows that the MapReduce implementation always outperforms the single-machine implementation in SoftTF-IDF and Flexi. 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 SoftTF-IDF and Flexi are so close that cannot be differentiated in the figure.

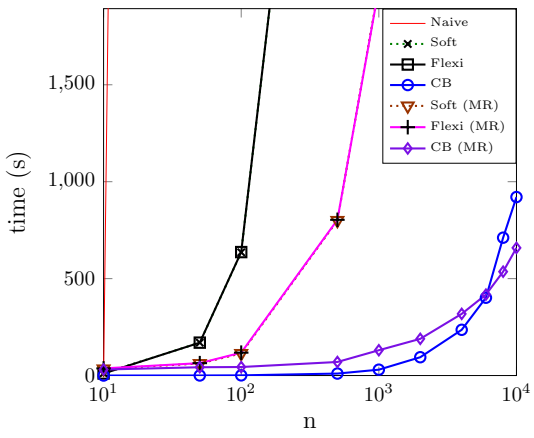


Figure 3.6: Response times of naive and different implementations of MC

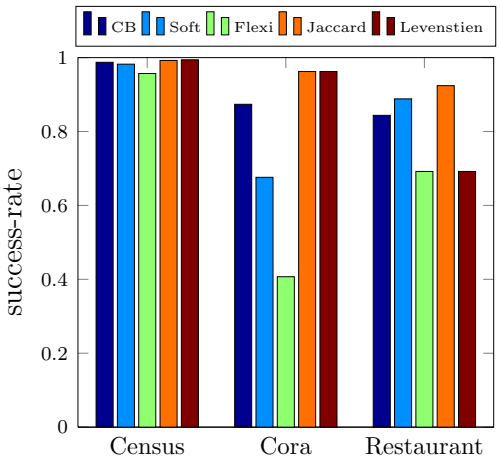


Figure 3.7: Success-rate of similarity functions over different datasets

Performance of CB

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

Figures 3.7 and 3.8 respectively compare the success-rate and maximum F1 of CB, SoftTF-IDF, Flexi, 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.

Figure 3.9 shows the 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 significant for the context-sensitive similarity functions (i.e. SoftTF-IDF and Flexi) over all datasets. This is one of the reasons that SoftTF-IDF and Flexi perform inefficiently in the MC algorithm.

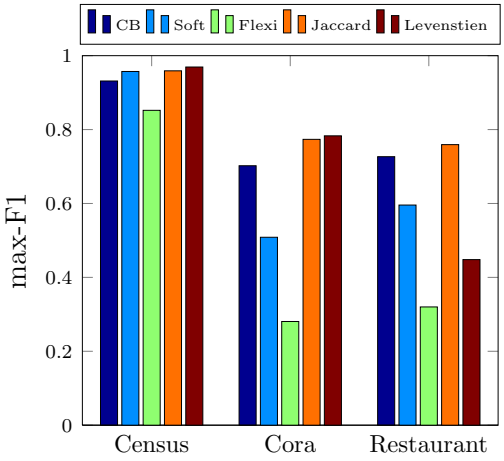


Figure 3.8: Max-F1 of similarity functions over different datasets

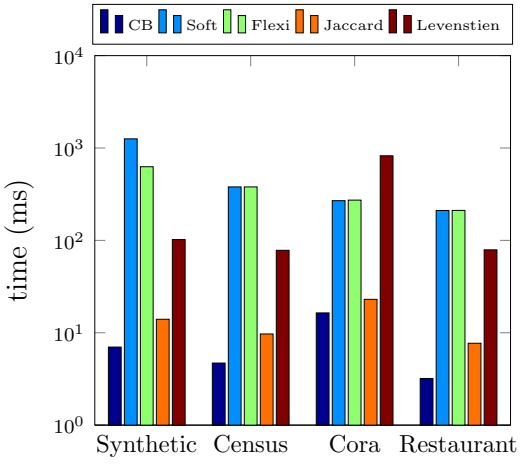


Figure 3.9: Average response time over different datasets

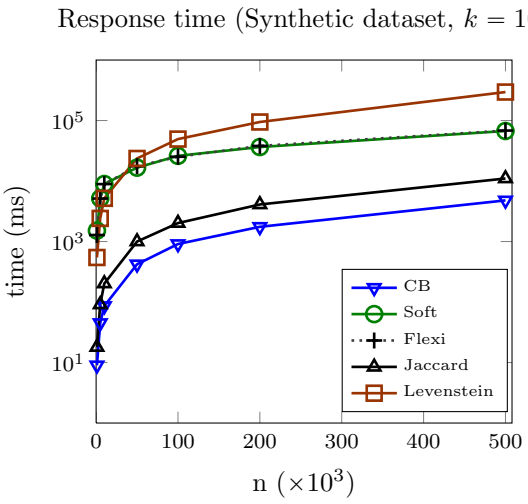


Figure 3.10: Response time of different similarity functions vs. n

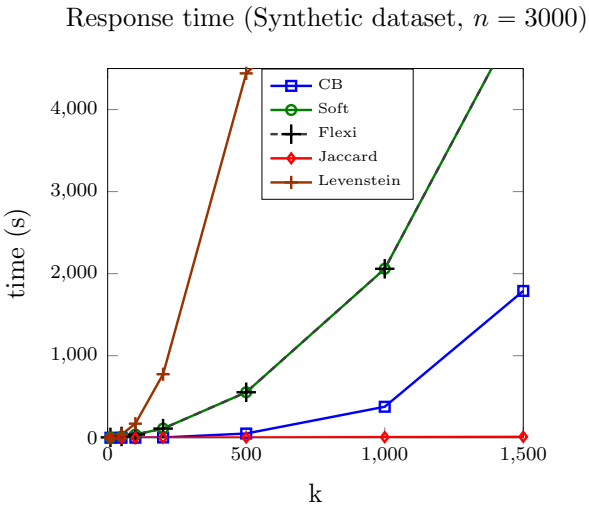


Figure 3.11: Response time of different similarity functions vs. k

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 3.10 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 3.11 shows the response time over Synthetic dataset with increasing k up to 1,500 and n set to 3,000. 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 SoftTF-IDF and Flexi 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 attribute values. 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.

3.6 Analysis against related work

Beyond the work on entity resolution presented in Chapter 2, nearest neighbor query processing, and top-k query processing over uncertain data are relevant to the ERPD problem.

In nearest neighbor query over uncertain data, given an uncertain query object q and the probability threshold τ, the aim is to find all uncertain objects in the database whose probability of being the nearest neighbor of q is higher than τ.

Cheng et al. [44] propose an approach for answering nearest neighbor queries in a moving object environment. They model each moving object in one of the following general ways: 1) an uncertainty region $U(t)$, which is a closed region that the object can only be found there in time t, and 2) an uncertainty density function $f(x, y, t)$, which is the probability density function of the object at location (x, y) and time t. In [44], the following four steps are performed for processing a nearest neighbor query: 1) Projection: using the last recorded location of the object, elapsed time, and the speed of the object, the uncertainty region of each object is computed; 2) Pruning: the lowest maximum distance, say d, to the query point q is computed and any object whose shortest distance to q is higher than d is pruned; 3) Bounding: drawing a bounding circle C centered at query point q with radius d, any portion of an object's uncertainty region that is outside C is ignored; and 4) Evaluation: integrating over different values of r, the probability of object o being the nearest neighbor of query point q is calculated

by computing the probability that o resides within the distance r from q times the probability that all other objects reside within a larger distance than r from q.

The nearest neighbor problem is similar to the ERPD problem in the sense that the qualification of an object, to be in the query result, depends on the attribute values and probabilities of the other objects in the database. The used pruning techniques in the nearest neighbor problem, however, are not applicable to the ERPD problem.

In a top-k query over uncertain data, given the query q, k, and the ranking function f, the aim is to find k tuples that have the highest probability to be the top-k tuples in the query result, where the query results are ranked according to ranking function f.

There have been some proposals dealing with uncertain top-k query processing (e.g., [52, 121, 142, 112]). The work in [121] extends the semantics of top-k queries from deterministic 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 [121] for defining the new semantics of entity resolution over probabilistic data. The proposal in [142] avoids enumerating the possible worlds space by using a dynamic programming approach for efficiently processing top-k queries on uncertain databases.

The top-k problem is similar to the ERPD problem in the sense that the qualification of a tuple cannot be computed independently from the other tuples in the database. Overall, the top-k proposals rely on a ranking function, which assigns a fixed unique rank to every uncertain tuple in the query result. Our problem setting, however, 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.

3.7 Conclusion

In this chapter, 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 matching pair and most-probable matching entity as the outcomes of ERPD. Then, we proposed CFA, a PTIME algorithm, which is applicable to the context-free similarity functions. The CFA algorithm is not applicable for context-sensitive similarity functions. Thus, we proposed an approximation algorithm, called MC, based on the Monte-Carlo method for

approximating the outcome of ERPD. To speedup the MC algorithm, we proposed a parallel version of it using the MapReduce framework.

To overcome the high response time of most context-sensitive similarity functions in the literature, which makes them very inefficient for our MC algorithm, we proposed the novel CB similarity function with the following salient features:

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

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

We showed the effectiveness of our proposed approaches using extensive experimentation over both synthetic and real datasets.

Chapter 4

Entity Resolution for Distributed Probabilistic Data[1]

4.1 Introduction

As discussed in Chapter 1, the concept of entity resolution (ER) is defined differently in different literature. In this chapter, we focus on the ER's definition related to *identity resolution* (see Definition 1.2.1).

In chapters 1 and 3, we discussed the problem of entity resolution over probabilistic data (ERPD), which arises in many applications that deal with probabilistic data. In many of these applications, probabilistic data is distributed among a number of nodes. Let us give two examples of such applications, one from image retrieval, and the other from scientific data management domains.

4.1.1. Example. *Matching images in facial image databases.* For investigation purposes, consider that the police keeps a database of facial images of all persons who leave the country. In order to gather such data, suppose that the police has installed supervenience video cameras in all ports including airports, seaports, and land border ports. In each port, video cameras capture the video of the persons who leave the country. A face recognition system is then used to extract each individual's facial image from captured videos, and store its feature vector as a tuple in a local database at the port. However, since the detected face might be moving in the video, and there is also an inherent uncertainty related to automated face recognition methods, each feature vector is associated with a probability value that represents its degree of certainty (e.g. as in [146]). Local databases are connected through a distributed system, which provides a query interface at each port for querying the set of all databases. At the same time, when a witness of a crime scene describes the facial features of a perpetrator, he/she is usually is not completely certain about some of the features. Therefore, the

[1]The material of this chapter has been partially published in [22] and [16].

police typically represents such a perpetrator's face using an uncertain entity, say e, consisting of a number of alternative feature vectors each of which is associated with a confidence value. An interesting question for the police is to identify if this perpetrator has left the country through any port, and thus querying the distributed database of all ports, in order to find the person who is most probably the same person as e, where $date > x$ (i.e. the date of the crime).

4.1.2. Example. *Finding astronomical objects in astrophysics data.* In astrophysics, as well as other scientific disciplines, the correlation and integration of the gathered observational data is the key for gaining new scientific insights. Astronomical observatories, distributed all over the world, produce data about sky surveys, some of which is uncertain [104]. In its simplified form, suppose each observatory maintains a single uncertain relation called *Objects*, which contains data about the observed *astronomical objects* in its sky surveys. Each *object* in such a relation is then represented using a number of alternative tuples, each with a *membership probability*, showing its degree of certainty. However, the alternatives are mutually exclusive, meaning that at most one of them can be true. The uncertainty model, which is used in this example, has been already used in the database literature for representing astrophysics data [122]. Astrophysics researchers who want to gather information about a particular astronomical object, supposedly represented by an uncertain entity e, are very interested in querying the astronomical objects observed in one sky's region, in all distributed observatories, in order to find the object which is most probably the same object as the given object e.

A straightforward approach for answering the above two queries is to ask all distributed nodes to send their data to a central node that can then deal with the problem of ERPD, by using one of the methods that are presented in Chapter 3. However, this approach is very expensive and does not scale well, neither with the size of databases, nor with the number of nodes. Therefore, using a distributed algorithm for dealing with the ERPD problem over distributed data is inevitable.

In this chapter, we propose the FD (Fully Distributed), which is a decentralized algorithm for dealing with the ERPD problem over distributed data, with the goal of minimizing the bandwidth usage and reducing the processing time. To the best of our knowledge, FD is the first proposal that deals with the ERPD problem over distributed data. It has the following salient features. First, it uses the novel concepts of *Potential* and *essential-set* to prune data at local nodes. This leads to a significant reduction of bandwidth usage compared to the baseline approaches. Second, its execution is completely distributed and does not depend on the existence of any certain node. We have validated FD through both implementation over a 75-node cluster and simulation. We have used both synthetic and real-world data in our experiments. The results show very good performance, in terms of bandwidth usage and response time.

The rest of the chapter is organized as follows. In Section 4.2, we make precise our assumptions, and formally define the problem. In Section 4.3, we present FD, and in Section 4.5, we analyze its communication cost. In Section 4.6, we report the performance evaluation of FD through implementation and simulation. Section 4.7 discusses analysis against related work, and Section 4.8 concludes.

4.2 Problem definition

In this section, we precisely define the problem addressed in the chapter.

For representing an uncertain database, we use the x-relation probabilistic data model [10] in which each uncertain entity is represented with an x-tuple (see the definition in Section 2.1.2). We denote an uncertain database by $\mathcal{D}$, the set of its possible worlds by $PW(\mathcal{D})$, and the set of all tuples in $\mathcal{D}$ by D.

We assume that the uncertain database is fragmented over a number of nodes in a distributed system. We make no specific assumption about the topology of the distributed system architecture, which can be very general, e.g. an unstructured P2P system or a cluster. In the distributed system, each node knows some other nodes, i.e. its neighbors, to communicate with.

We define the problem of entity resolution for distributed probabilistic data as follows. Let e be an uncertain entity issued at a query originator p. Let TTL (Time To Live) determine the maximum hop distance which the user wants the entity resolution message to travel. Let $\mathcal{D}$ be the union of the uncertain databases that are in the schema of e, and maintained by nodes that can be accessed through TTL hops from the query originator. Let Sim be a context-free similarity function (refer to Section 3.2.2 for definition) for computing the similarity between tuples. Our goal is to find the most probable matching pair of e and $\mathcal{D}$, i.e. MPMP(e, $\mathcal{D}$) (see Definition 3.2.2), while minimizing the communication cost.

4.3 Distributed Computation of Most-probable Matching-pair

One possible approach for computing MPMP is to move all relevant data of nodes to a central node, e.g. the query originator, where MPMP is computed using a centralized algorithm. However, the problem with this approach is that the query originator becomes a *communication bottleneck*, since it must receive a large amount of data from other nodes. In addition, it becomes a *processing bottleneck*, as it must process a large amount of data. In this section, we propose a fully distributed algorithm called FD, for computing MPMP. Our algorithm avoids the problems of the centralized approach by : 1) pruning all data that has

no chance to be MPMP, thus reducing the communication cost significantly; and 2) distributing the processing of MPMP over a large number of nodes.

4.3.1 Algorithm Overview

The FD algorithm starts at the query originator, the node at which a user issues a query involving an uncertain entity e to be resolved. The query originator performs some initialization. First, it sets TTL to a value which is either specified by the user or default, as found sufficient by the system in previous calculations. Second, it gives e a unique identifier, denoted by *eid*, which is made of a unique node-ID and a query counter managed by the query originator. Nodes use *eid* to distinguish between new queries and those received before. After initialization, e is included in a message that is then broadcasted by the query originator to its reachable neighbors. Next, the entity resolution proceeds in the following phases done at each node that receives the query:

- **Query forward.** Each node p that receives the message including e from a node q performs the following steps. If it is the first time of receiving the query, then the node p saves the id of q as its parent, else discards the message and makes a new message including *eid* and sends it to q to indicate that the query has been received from another node. Then p decrements TTL by one, if $TTL > 0$, it makes a new message including e, *eid*, and new TTL; sends the message to all neighbors except q; and saves the number of sent messages to the neighbors.

- **Extract the essential-set.** The core idea of this phase is that for computing the most probable matching pair, the query originator does not need all entity-tuple pairs maintained at p, but only a subset of them that we call *essential-set*. In this phase, p extracts the essential-set of its local data and saves it locally until receiving the essential-sets of its neighbors to which it has sent the query.

- **Merge-and-backward essential-sets.** In this phase, p unifies its essential-set with those received from its neighbors into a set of entity-tuple pairs $essential_{pq}$, and sends $essential_{pq}$ to its parent, the node from which it received the query.

- **MPMP computation and data retrieval.** During the first three phases of the algorithm, the query originator receives a number of merged essential-sets from its neighbors. It unifies these sets with its local essential-set into the set $essential_{unified}$, and computes MPMP(e, $\mathcal{D}$) and asks the node which contains the data to return the data content.

In the next subsections, we describe in more details the FD algorithm phases.

4.3.2 Extract the Essential-set

At each node p, our FD algorithm prunes the data that have no chance to be the (global) most probable matching pair, i.e. MPMP(e, $\mathcal{D}$). For this, FD needs to extract the essential-set of each node which we define as follows. Let e be the given entity. Suppose $\mathcal{D}_p$ is the database maintained by p and n_p is the number of tuples in D_p. Let S_p be the set of all entity-tuple pairs at p, i.e. $S_p = e \times D_p$. We define the *essential-set* of S_p, denoted by $essential(S_p)$ by using its complement: $essential^c(S_p)$ is a subset of S_p whose members can never be MPMP(e, $\mathcal{D}$).

The alternatives of e are mutually exclusive, thus to find $essential(S_p)$, it is sufficient to compute the essential-set for each alternative $t \in e$, and then unify the essential-sets of all alternatives of e. More precisely, we have:

$$essential(S_p) = \bigcup_{t \in e} essential(S_{p,t}), \text{where } S_{p,t} = \{t\} \times D_p$$

Now, we consider an alternative $t \in e$, and explain the process of finding $essential(S_{p,t})$.

Let $L_p = \{(t, t_{p,1}), \ldots, (t, t_{p,n_p})\}$ be the list of $S_{p,t}$ pairs sorted in decreasing order of the similarities between t and D_p tuples. In other words, we have:

$$Sim(t, t_{p,1}) > \ldots > Sim(t, t_{p,n_p}),$$

where Sim is the given similarity function.

In the FD algorithm, we need to merge entity-tuple pair lists from other nodes with the pairs in list L_p. Let $\rho = (t, t_q)$ be an entity-tuple pair, from a node other than p, that should be merged with L_p. The pair ρ may be inserted in any index of L_p, say index $i \in [1, n_p + 1]$, based on the similarity between t and t_q. The question in pruning is whether this pair has any chance to be $MPMP(e, \mathcal{D})$ or not. The answer to this question depends on the value of $P_{msp}(\rho, \mathcal{D})$, i.e. the probability that the pair ρ is the most similar pair[2] (see Definition 3.2.2 in Section 3.2.3). However, $P_{msp}(\rho, \mathcal{D})$ depends not only on the pairs that are at node p, but also on the pairs of other nodes. Thus, we cannot compute the exact value of $P_{msp}(\rho, \mathcal{D})$ locally, but we can compute an upper bound on this value. We denote such upper bound as the *Potential* of the index i of list L_p. More precisely,

$$Potential(i) = max P_{msp}(\rho, \mathcal{D}) \tag{4.1}$$

where $i \in [1..n_p + 1]$.

For instance, $Potential(1)$ is the maximum possible (global) value for P_{msp} of the pair that is inserted in the first location of list L_p. The following lemma computes the Potential of index i of list L_p.

[2]Notice that $P_{msp}(\rho, \mathcal{D})$ is the global P_{msp} value of pair ρ, while $P_{msp}(\rho, \mathcal{D}_p)$ is its local P_{msp} value at node p. Generally, $P_{msp}(\rho, \mathcal{D}_p) \geqslant P_{msp}(\rho, \mathcal{D})$.

4.3.1. Lemma. *Let i be an index in range $[1..n_p+1]$. Let Y be the set of x-tuples formed by considering correlations between the tuples $\{t_{p,1}, \ldots, t_{p,i-1}\}$, then*

$$Potential\,(i) = P(t) \times \prod_{x \in Y}(1 - P(x)).$$

Proof. Let $S_t = t \times D$ and $L = \{(t, t_1), \ldots, (t, t_n)\}$ be the list of S_t pairs sorted based on their similarity in descending order. Let $\rho = (t, t_q)$ resides in the index j of list L, i.e. $L[j] = \rho$. Using equation (3.1) (refer to Section 3.3, we have

$$P_{msp}(\rho, \mathcal{D}) = P(t_q) \times P(t) \times \prod_{x \in X}(1 - P(x)) \tag{4.2}$$

where X is the set of x-tuples formed by considering correlations between the tuples t_1 to t_j while the x-tuple containing t_q is omitted from it. It is clear that the value of $P(t_q)$ which maximizes RHS(4.2) is equal to one. The set of tuples t_1 to t_{j-1} can be partitioned into two sets T_1 and T_2, where $T_1 = \{t_{p,1}, \ldots, t_{p,i-1}\}$ is a subset of D_p and T_2 is a subset of $D - D_p$. Let X_1 and X_2 be the set of x-tuples formed by considering correlations between the tuples in T_1 and T_2, respectively. Since all members of an x-tuple reside within the same node, x-tuple set X in RHS(4.2) can be partitioned into two disjoint sets $X = X_1$ and X_2, and, setting $P(t_q)$ to one, equation (4.2) can be rewritten as

$$P_{msp}(\rho, \mathcal{D}) = P(t) \times \prod_{x \in X_1}(1 - P(x)) \times \prod_{x \in X_2}(1 - P(x)) \tag{4.3}$$

Notice that since we set $P(t_q)$ to one, no x-tuple can contain it. Set X_1 is fixed, but we can make any assumption about set X_2 to maximize RHS(4.3). Each x-tuple x in set X_2 reduces RHS(4.3) by the factor of $1 - P(x)$, thus, RHS(4.3) is maximized when $X_2 = \varnothing$. In such case, RHS(4.3) is equal to the asserted value in the lemma. □

4.3.2. Corollary. *Potential is a monotonically decreasing function.*

Intuitively, Corollary 4.3.2 says that the higher is the index, the lower is its potential.

Let *local_max* be the maximum local P_{msp} value of pairs in list L_p, i.e. $local\_max = max P_{msp}(\rho, \mathcal{D}_p), \rho \in L_p$. We use *local_max* to define the *stop* index of list L_p as the smallest index in $[1..n_p + 1]$ where

$$Potential(stop) < local\_max. \tag{4.4}$$

The following lemma provides the basis for pruning the pairs in list L_p.

4.3.3. Lemma. *Let stop be the stop index of list L_p. Then,*

$$\forall i \in [stop, n_p], L_p[i] \neq \arg\max_{\rho \in L_p} P_{msp}(\rho, \mathcal{D}).$$

Proof. Let j be the index of a pair in list L_p with maximum local P_{msp} value, i.e. $P_{msp}(L_p[j], \mathcal{D}_p) = local\_max$. Let i be an index in list L_p, where $i \in [stop, n_p]$. Let $S_t = \{t\} \times D$ and $L = \{(t, t_1), \ldots, (t, t_n)\}$ be the list of S_t pairs sorted based on their similarity in descending order. Let j' and i' respectively be the index of pairs $L_p[j]$ and $L_p[i]$ in list L, i.e. $L[j'] = L_p[j]$ and $L[i'] = L_p[i]$. To prove the lemma, we show that

$$P_{msp}(L[i'], \mathcal{D}) < P_{msp}(L[j'], \mathcal{D}) \tag{4.5}$$

We have

$$P_{msp}(L[i'], \mathcal{D}) = P(t) \times P(t_{i'}) \times \prod_{x \in X_{i'}} (1 - P(x)) \tag{4.6}$$

where $X_{i'}$ is the set of x-tuples formed by considering correlations between the tuples t_1 to $t_{i'}$ while the x-tuple containing $t_{i'}$ is omitted from it. The set of tuples t_1 to $t_{i'}$ can be partitioned into two sets $T_{i',1}$ and $T_{i',2}$, where $T_{i',1} = \{t_{p,1}, \ldots, t_{p,i-1}\}$ is a subset of D_p and $T_{i',2}$ is a subset of $D - D_p$. Let $X_{i',1}$ and $X_{i',2}$ respectively be the set of x-tuples formed by considering correlations between the tuples in $T_{i',1}$ and $T_{i',2}$, while the x-tuple containing $t_{i'}$ is omitted from $X_{i',1}$. Since all members of an x-tuple reside within the same node, x-tuple set $X_{i'}$ in RHS(4.6) can be partitioned into two disjoint sets $X_{i',1}$ and $X_{i',2}$, and equation (4.6) can be rewritten as

$$P_{msp}(L[i'], \mathcal{D}) = P(t) \times P(t_{i'}) \times \prod_{x \in X_{i',1}} (1 - P(x)) \times \prod_{x \in X_{i',2}} (1 - P(x)) \tag{4.7}$$

Since $L[i']$, $L_p[i]$, $(t, t_{i'})$, and $(t, t_{p,i})$ refer to the same pair, equation (4.7) can be written as

$$P_{msp}(L[i'], \mathcal{D}) = P_{msp}(L_p[i], \mathcal{D}_p) \times \prod_{x \in X_{i',2}} (1 - P(x)) \tag{4.8}$$

Using the same notation, we can write $P_{msp}(L[j'], \mathcal{D})$ as

$$P_{msp}(L[j'], \mathcal{D}) = P_{msp}(L_p[j], \mathcal{D}_p) \times \prod_{x \in X_{j',2}} (1 - P(x)) \tag{4.9}$$

Based on the definition of stop index, it is clear that $stop > j$, thus yielding $i > j$. Thus, $i' > j'$ and we have

$$(\forall x \in X_{j',2}, \exists y \in X_{i',2} \mid x \subseteq y) \Rightarrow \prod_{y \in X_{i',2}} (1 - P(x)) \leqslant \prod_{x \in X_{j',2}} (1 - P(y)) \tag{4.10}$$

Moreover, we know that

$$P_{msp}(L_p[j], \mathcal{D}_p) = local\_max > P_{msp}(L_p[i], \mathcal{D}_p) \tag{4.11}$$

Using (4.8), (4.9), (4.10) and (4.11), we have

$$P_{msp}(L[i'], \mathcal{D}) < P_{msp}(L[j'], \mathcal{D}) \tag{4.12}$$

Since $L[i'] = L_p[i]$ and $L[j'] = L_p[j]$, (4.12) implies that

$$L_p[i] \neq \arg\max_{\rho \in L_p} P_{msp}(\rho, \mathcal{D}).$$

□

Intuitively, Lemma 4.3.3 says that among all pairs in list L_p, one of the pairs before the stop index has the maximum global P_{msp} value.

4.3.4. Corollary. *Let stop be the stop index of list L_p. Then,*

$$\forall i \in [stop, n_p], L_p[i] \neq MPMP(e, \mathcal{D}).$$

Intuitively, Corollary 4.3.4 says that the pair at the stop index and any pair after it have no chance to be the most probable matching pair. Thus, $essential(S_{p,t})$ is the set of L_p pairs whose index is smaller than the stop index.

Algorithm

Algorithm 6 describes the details of the steps which are performed for finding $essential(S_p)$. Steps 3-19 are repeated for every alternative of e, say t, and at each iteration compute $essential(S_{p,t})$. Step 3 computes set $S_{p,t}$ and step 4 sorts its pairs based on the similarity between the pair elements in descending order according to similarity function Sim, and stores the result in list L. Steps 5-7 compute list T as the second elements of list L and do some initialization. Steps 9-16 are repeated until finding the *stop* index of L, and in each iteration, they process the pair at index i of list L. Steps 10-12 compute $P_{msp}(L[i], \mathcal{D}_p)$ as the intersection of two independent probabilistic events: t occurs; and among tuples $T[1]$ to $T[i]$, only $T[i]$ occurs. To calculate the probability of the latter event, step 10 considers correlation among tuples to group tuples $T[1]$ to $T[i]$ into the set of x-tuples Y and step 11 removes the x-tuple containing $T[i]$ from Y and stores the result in x-tuple set X. Steps 13-15 update the current maximum P_{msp} of the pairs which we have processed so far. Steps 16 computes $Potential(i+1)$ using the x-tuple set Y which has already been computed in step 10. Step 17 checks if all pairs in the list L have been processed or $i+1$ is the *stop* index of L. If the condition holds, then the algorithm stops processing list L, else it continues by processing the next pair in L. Step 19 adds pairs $L[1]$ to $L[stop-1]$ to the *essential-set*. Finally, step 21 returns the *essential-set*.

Algorithm 6 finding the *essential-set*

Input:
-Entity e
-Database $\mathcal{D}_p$
-Similarity function Sim

Output: $essential(S_p)$, where $S_p = e \times D_p$

1: $essential \leftarrow \varnothing$
2: **for all** $t \in e$ **do**
3: $S_{p,t} \leftarrow \{t\} \times D_p$
4: $L \leftarrow Sort(S_{p,t}, Sim)$
5: $T \leftarrow \{t' \mid (t, t') \in L\}$
6: $local\_max \leftarrow -1$
7: $i \leftarrow 0$
8: **repeat**
9: $i \leftarrow i + 1$
10: $Y \leftarrow$ set of x-tuples involved in $\{T[1], \ldots, T[i]\}$
// removing the x-tuple containing $T[i]$
11: $X \leftarrow Y - \{x \mid x \in Y \wedge T[i] \in x\}$
12: $P_{msp} \leftarrow P(t) \times P(T[i]) \times \prod_{x \in X}(1 - P(x))$
13: **if** $P_{msp} > local\_max$ **then**
14: $local\_max \leftarrow P_{msp}$
15: **end if**
16: $Potential \leftarrow P(t) \times \prod_{x \in Y}(1 - P(x))$
17: **until** $(Potential < local\_max) \vee (i = |L|)$
18: $stop \leftarrow i + 1$
19: $essential \leftarrow essential \cup \{L[1], \ldots, L[stop - 1]\}$
20: **end for**
21: return $essential$

Example

Let us illustrate the process of extracting essential-set using an example. Consider the uncertain entity e and the uncertain database $\mathcal{D}_p$ (maintained at node p) shown in Figures 4.1(a) and 4.1(b) respectively. In this example, D_p contains single-alternative x-tuples, and entity e has only one alternative. The set of existing entity-tuple pairs in node p, i.e. set S_p, can be computed as $S_p = e \times D_p$. To prune S_p, we sort its pairs based on their similarity in descending order. The resulted list, denoted by L, is shown in Figure 4.1(c).

The $Potential$ of the first location in list L, i.e. $Potential(1)$, is equal to the probability of the event that t occurs, which is equal to $P(t) = 0.8$. The P_{msp} of the first entity-tuple pair in L, i.e. $(t, t_{p,3})$, is equal to the probability of the event that t and $t_{p,3}$ occur, thus, $P_{msp}\left((t, t_{p,3}), \mathcal{D}_p\right)$ is equal to $P(t) \times P(t_{p,3}) = 0.08$. The $Potential$ of the second location in list L, i.e. $Potential(2)$, is equal to the

t	P(t)
t	0.8

(a) e

t	P(t)
$t_{p,1}$	0.4
$t_{p,2}$	0.7
$t_{p,3}$	0.1
$t_{p,4}$	0.2
$t_{p,5}$	0.9
$t_{p,6}$	0.8
$t_{p,7}$	0.7
$t_{p,8}$	0.9

(b) $\mathcal{D}_p$

i	L[i]	$\mathbf{P_{msp}(L[i], \mathcal{D}_p)}$	Potential(i)
1	$(t, t_{p,3})$	0.08	0.8
2	$(t, t_{p,7})$	0.504	0.72
3	$(t, t_{p,8})$	0.1944	0.216
4	$(t, t_{p,1})$	0.00864	0.0216
5	$(t, t_{p,2})$	0.00907	0.01296
6	$(t, t_{p,4})$	0.00078	0.00389
7	$(t, t_{p,5})$	0.00280	0.00311
8	$(t, t_{p,6})$	0.00025	0.00031
9	-	-	0.000062

(c) List L

Figure 4.1: An example of uncertain entity e, database $\mathcal{D}_p$, and the pruning process

probability of the event that t occurs but $t_{p,3}$ does not occur, which is equal to $P(t) \times (1 - P(t_{p,3})) = 0.72$. This means that the maximum possible value for P_{msp} of an entity-tuple pair which comes in $L[2]$ is 0.72. Since the *Potential* is greater than the current maximum value of P_{msp}, i.e. 0.08, we continue processing the list.

The P_{msp} of the second pair in L, i.e. $(t, t_{p,7})$, is equal to the probability of the event that t and $t_{p,7}$ occur but $t_{p,3}$ does not occur, which is equal to $P(t) \times P(t_{p,7}) \times (1 - P(t_{p,3})) = 0.504$. The *Potential* of the third location in list L, i.e. $Potential(3)$, is equal to the probability of the event that t occurs but neither $t_{p,3}$ nor $t_{p,7}$ occurs, which is equal to $P(t) \times (1 - P(t_{p,3})) \times (1 - P(t_{p,7})) = 0.216$. At this point, we stop processing the list since the *Potential* is less than the current maximum value of P_{msp}, i.e. 0.504. Therefore, the *stop* index of L_p is 3 and $essential(S_p)$ is equal to $\{(t, t_{p,3}), (t, t_{p,7})\}$. To provide better intuition, the P_{msp} and *Potential* values for other pairs are also shown in Figure 4.1(c).

4.3.3 Merge-and-Backward Essential-Sets

After extracting its essential-set, each node p waits for receiving the essential-sets of its children (the nodes to which p has sent the query). After receiving the essential-set of its children (or after a default wait time), p merges its essential-set with those received from its children into a set of entity-tuple pairs $essential_{pq}$, and sends it to its parent.

In order to minimize network traffic, nodes do not bubble up the data items of entity-tuple pairs (which could be large), but only some needed information about them. The information that is put in the sent essential-set for each entity-tuple pair $(t_i, t_j), t_i \in e, t_j \in D_q$, is a vector (i, a, j, x, s, p) where i is the index of t_i in e, a is the address of node q which owns t_j, j is the index of tuple t_j in the database D_q maintained by q, x is the x-tuple to which t_j belongs, s is the similarity score between t_i and t_j, and p is the probability of tuple t_j.

4.3.4 MPMP Computation and Data Retrieval

When the query originator receives its children's essential-sets, it merges them with its local essential-set into the set $essential_{unified}$. Theorem 4.3.5 shows that $essential_{unified}$ contains all entity-tuple pairs which are needed for computing MPMP(e, $\mathcal{D}$).

4.3.5. THEOREM. *The entity-tuple pairs in set $essential_{unified}$ are sufficient for computing MPMP(e, $\mathcal{D}$).*

Proof. Let S be the set of all entity-tuple pairs at nodes which receive the query, i.e. $S = e \times D$. We show that we do not need any entity-tuple pair $\rho = (t, t'), \rho \in S - essential_{unified}$ for computing MPMP(e, $\mathcal{D}$).

Let L be the list of pairs in set $essential_{unified}$ which have alternative $t \in e$ as their first element, and sorted based on their similarity in descending order. Let $stop_p$ be the stop index of a node, say node p, which comes before the stop indices of other nodes in list L. Using Lemma 4.3.3, pairs which come at or after $stop_p$ in L, cannot be the pair of L with maximum P_{msp}. Thus, the pair of L with maximum P_{msp} lies in the range $[1..stop_p - 1]$. Now, we show that there is no entity-tuple pair $\rho = (t, t'), \rho \in S - essential_{unified}$, which may come before *stop* in list L, and thus, is needed for computing the pair of L with maximum P_{msp}. Pair ρ is either maintained at node p or at a node other than p, say q. In the former case, ρ comes after $stop_p$ in list L since $stop_p$ is the stop index of node p. Also in the latter case, ρ comes after $stop_p$ in list L since ρ comes after the stop index of q which itself comes after $stop_p$ in list L. Thus, using the pairs $L[1]$ to $L[stop - 1]$, we can compute the pair with maximum P_{msp} and thereby MPMP(e, $\mathcal{D}$). □

Algorithm 7 shows the detailed steps which the query originator performs to compute MPMP(e, $\mathcal{D}$). Notice that:

- *current_max* does not represent the maximum value of P_{msp} of the pairs in one list, i.e. related to alternative $t \in e$, but the current maximum P_{msp} value of the pairs which we have visited so far. Thus, we reset it only once in the beginning of the algorithm.

- We use *Potential* to stop early in visiting the pairs of list L. Notice that we may discard a list of pairs altogether because the maximum possible P_{msp} of the pairs in that list (i.e. $Potential(1)$) is less than the current maximum P_{msp} value that we got so far.

- Since the set S_t consists of a number of sorted lists, the sort function in step 8 uses the sort-merge algorithm to merge these sorted lists.

Algorithm 7 computing MPMP(e, $\mathcal{D}$)

Input: Set of entity-tuple pairs $essential_{unified}$
Output: MPMP(e, $\mathcal{D}$)

```
1: current_max ← −1
2: for all t ∈ e do
3:     Potential ← P(t)
4:     S_t ← {(t, t') | (t, t') ∈ essential_unified}
5:     length ← |S_t|
6:     i ← 1
7:     while (Potential > current_max) ∧ (i ⩽ length) do
8:         L ← Sort S_t pairs based on their similarity
9:         T ← {t' | (t, t') ∈ L}
10:        Y ← set of x-tuples involved in {T[1], ..., T[i]}
11:        X ← Y − {x | x ∈ Y ∧ T[i] ∈ x}
12:        P_msp ← P(t) × P(T[i]) × ∏_{x∈X}(1 − P(x))
13:        if P_msp > current_max then
14:            current_max ← P_msp
15:            MPMP ← L[i]
16:        end if
17:        Potential ← P(t) × ∏_{x∈Y}(1 − P(x))
18:        i ← i + 1
19:    end while
20: end for
21: return MPMP
```

Using Algorithm 7, the query originator computes MPMP(e, $\mathcal{D}$) and asks the node which contains it to return the data content which is then returned to the user.

In the next section, we provide an example of all phases of the FD algorithm.

4.4 FD Example

In this section, we illustrate the FD algorithm with an example.

Consider a network consisting of three nodes o, q, and p as shown in Figure 4.2(a) and let these nodes respectively contain $\mathcal{D}_O$, $\mathcal{D}_q$, and $\mathcal{D}_p$ single-alternative databases which are shown in Figures 4.2(c), 4.2(d), and 4.2(e), respectively. Suppose that the user submits entity e, shown in Figure 4.2(b), to the node o, then FD performs the following phases for computing MPMP(e, $\mathcal{D}$), where $\mathcal{D} = \mathcal{D}_O \cup \mathcal{D}_q \cup \mathcal{D}_p$:

1. In the *"query forward"* phase, node o forwards e to node q, and node q forwards e to node p.

2. In the *"extract the essential-set"* phase, each node extracts its essential-set as shown in Figures 4.2(h), 4.2(g), and 4.2(f). In the figures, we abbreviate *Potential* as *Po*. Also, the pairs that are transferred to the essential-set, are shown in bold, and since FD stops at the stop index, the values of P_{msp} and *Potential* for the pairs after the stop index are not shown. Notice that the stop index of list L_q if equal to $|L_q| + 1$, thus all pairs in this list are transferred into the essential-set.

3. In the *"merge-and-backward essential-sets"* phase, node p sends its essential set, i.e. $essential(S_p)$, to node q. Then, q merges the received set with its own essential set, i.e. $essential(S_q)$, into set $essential_{pq}$ (shown in Figure 4.2(i)), and sends it to node o.

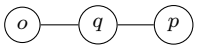
4. In the *"MPMP computation and data retrieval"* phase, node o merges its essential set, i.e. $essential_O$, with the received set from q, i.e. $essential_{pq}$, into set $essential_{unified}$ whose members are shown in Figure 4.2(j). Then, node o computes the MPMP as shown in Figure 4.2(j). The computed MPMP is equal to $(t, t_{p,4})$, thus node o asks node p for the data of the tuple $t_{p,4}$.

4.5 Analysis of Communication Cost

In this section, we analyze the communication cost of FD, and as we will see, it is relatively low. We measure the communication cost in terms of number of messages and number of bytes which should be transferred over the network in order to execute a query by our algorithm. The messages transferred can be classified as: (1) forward messages, for forwarding the query to nodes; (2) backward messages, for returning the essential-sets from nodes to the query originator; (3) retrieve message, to request and retrieve the MPMP. Let us first formalize the distributed system model that we use in our analysis.

4.5.1 Distributed System Model

Let P be the set of the nodes in the distributed system. Let Q be an entity resolution query at the query originator p_o, i.e. the node at which the query is issued. Let $P_Q \subseteq P$ be a set containing the query originator and all nodes that receive Q. We model the nodes in P_Q and the links between them by a graph $G(P_Q, E)$ where P_Q is the set of vertices in G and E is the set of the edges. There is an edge $p - q$ in E if and only if there is a link between the nodes p and q in the distributed system. Two nodes are called neighbor, if and only if there is an edge between them in G. The number of neighbors of each node $p \in P_Q$ is called the degree of p and is denoted by $d(p)$.



(a) network

t	P(t)
t	0.3

(b) e

t	P(t)
$t_{O,1}$	0.6
$t_{O,2}$	0.4
$t_{O,3}$	0.3
$t_{O,4}$	0.7
$t_{O,5}$	0.3
$t_{O,6}$	0.5

(c) $\mathcal{D}_O$

t	P(t)
$t_{q,1}$	0.2
$t_{q,2}$	0.1
$t_{q,3}$	0.2
$t_{q,4}$	0.2
$t_{q,5}$	0.1
$t_{q,6}$	0.1

(d) $\mathcal{D}_q$

t	P(t)
$t_{p,1}$	0.2
$t_{p,2}$	0.9
$t_{p,3}$	0.8
$t_{p,4}$	0.8
$t_{p,5}$	0.7
$t_{p,6}$	0.9

(e) $\mathcal{D}_p$

Pair	$\mathbf{P_{msp}}$	Po
$(\boldsymbol{t, t_{O,5}})$	0.09	0.3
$(\boldsymbol{t, t_{O,2}})$	0.084	0.21
$(\boldsymbol{t, t_{O,1}})$	0.08	0.13
$(t, t_{O,4})$	-	0.05
$(t, t_{O,3})$	-	-
$(t, t_{O,6})$	-	-

(f) L_O

Pair	$\mathbf{P_{msp}}$	Po
$(\boldsymbol{t, t_{q,3}})$	0.06	0.3
$(\boldsymbol{t, t_{q,6}})$	0.02	0.24
$(\boldsymbol{t, t_{q,4}})$	0.04	0.22
$(\boldsymbol{t, t_{q,1}})$	0.03	0.17
$(\boldsymbol{t, t_{q,5}})$	0.01	0.14
$(\boldsymbol{t, t_{q,2}})$	0.01	0.12
-	-	0.11

(g) L_q

Pair	$\mathbf{P_{msp}}$	Po
$(\boldsymbol{t, t_{p,4}})$	0.24	0.3
$(t, t_{p,6})$	-	0.06
$(t, t_{p,1})$	-	-
$(t, t_{p,5})$	-	-
$(t, t_{p,3})$	-	-
$(t, t_{p,2})$	-	-

(h) L_p

Pair
$(t, t_{q,3})$
$(t, t_{p,4})$
$(t, t_{q,6})$
$(t, t_{q,4})$
$(t, t_{q,1})$
$(t, t_{q,5})$
$(t, t_{q,2})$

(i) $\text{essential}_{\text{pq}}$

Pair	$\mathbf{P_{msp}}$	Po
$(t, t_{q,3})$	0.06	0.3
$(t, t_{p,4})$	0.19	0.24
$(t, t_{O,5})$	-	0.05
$(t, t_{q,6})$	-	-
$(t, t_{q,4})$	-	-
$(t, t_{O,2})$	-	-
$(t, t_{q,1})$	-	-
$(t, t_{q,5})$	-	-
$(t, t_{q,2})$	-	-
$(t, t_{O,1})$	-	-

(j) $\text{essential}_{\text{unified}}$

Figure 4.2: Illustration of different phases of the FD algorithm using a simple example

A peer $p \in P_Q$ may receive Q from some of its neighbors. The first node, say q, from which p receives Q, is the parent of p in G, so p is a child of q. A node may have some neighbors that are neither its parent nor its children.

4.5.2 Forward Messages

Forward messages are the messages that we use to forward Q to the nodes. According to the basic design of our algorithm, each node in P_Q sends Q to all its neighbors except its parent. Let p_o denote the query originator. Let $G(P_Q, E)$ be a graph representing the distributed network, such that P_Q is the set of nodes and E is the set of links between the nodes. With our FD algorithm, each node $p \in \{P_Q - \{p_o\}\}$, sends Q to $d(p)-1$ nodes, where $d(p)$ is the degree of p in G. The query originator sends Q to all of its neighbors, in other words to $d(p_o)$ nodes. Then, the sum of all forward messages m_{fw} can be computed as

$$m_{fw} = d(p_o) + \sum_{p \in \{P_Q - \{p_o\}\}} (d(p) - 1)$$

We can write m_{fw} as follows:

$$m_{fw} = \left(\sum_{p \in \{P_Q\}} (d(p) - 1) \right) + 1 = \left(\sum_{p \in \{P_Q\}} d(p) \right) - |P_Q| + 1 \tag{4.13}$$

We use the average degree of the graph G, denoted by $d(G)$, to simplify (4.13). $d(G)$ is defined as the average degree of nodes in G and can be computed as

$$d(G) = \frac{\sum_{p \in P_Q} d(p)}{|P_Q|}$$

Substituting $d(G)$ in (4.13), we have

$$m_{fw} = (d(G) - 1) \times |P_Q| + 1$$

From the above discussion, we can derive the following Lemma.

4.5.1. Lemma. *The number of forward messages in the FD algorithm is* $(d(G) - 1) \times |P_Q| + 1$.

Proof. Implied by the above discussion. □

In our underlying applications, e.g. astronomy application, the average degree of nodes is low, that is each node is usually connected to a small number of other nodes. Thus, the total number of forward messages is not very high compared to the number of nodes. For example, if the average degree of the system is 4, i.e. $d(G) = 4$, then we have $m_{fw} = 3 \times |P_Q| + 1$.

Let b_t be the average size of a tuple in Q in bytes, and $|Q|$ be the number of alternative tuples of Q. Then, the total size of data transferred by forward messages, denoted by b_{fw}, can be computed as $((d(G) - 1) \times |P_Q| + 1) \times |Q| \times b_t$.

4.5.3 Backward Messages

In the Merge-and-Backward phase, each node in P_Q, except the query originator, sends its merged essential-set to its parent. Therefore, the number of backward messages, denoted by m_{bw}, is $m_{bw} = |P_Q| - 1$.

In the query forward phase of the algorithm, nodes in P_Q are arranged in a tree, called *query-tree*, with the query originator as its root. For our modeling, we assume that the query-tree is a k-ary tree (i.e. $k = d(G)$) in which the root has k children and all intermediate nodes has exactly $k - 1$ children. Moreover, we assume that all leaves are at the same level. These assumptions, however, are mostly for illustration purposes. In practice, nodes are organized in arbitrary tree topologies.

Let h be the height of the tree, with the root at level $l = 0$. The total number of nodes, i.e. $|P_Q|$, can be computed as $|P_Q| = (\sum_{l=2}^{h}(k-1)^l) + k + 1$.

Let $S(l)$ be the total size of data transferred in the Merge-and-Backward phase by each node which resides in the level l of the query-tree. Let b_{es} be the average size of the essential-set of a node. In the Merge-and-Backward phase, each node at level h of the query-tree, i.e. leaf nodes, sends its essential-set to its parent. Thus, $S(h) = b_{es}$. Also, each intermediate node at level l, $l \neq 0$, of the query-tree receives exactly $k - 1$ essential-sets from its children which reside at level $l + 1$; merges them with its essential-set; and send the merged essential-set to its parent. Thus, for each intermediate node we have $S(l) = (k-1) \times S(l+1) + b_{es}$; thereby yielding the following recurrence relation for $S(l)$:

$$S(l) = \begin{cases} (k-1) \times S(l+1) + b_{es} & \text{for } 0 < l < h \\ b_{es} & \text{for } l = h \end{cases}$$

By solving this recurrence relation, we have

$$S(l) = \frac{1 - (k-1)^{h-l+1}}{1 - (k-1)} \tag{4.14}$$

Since there are exactly $k \times (k-1)^{l-1}$ nodes at level l, $0 < l \leqslant h$, thus the total data transfer of the Merge-and-Backward phase, denoted by b_{bw}, can be computed as:

$$b_{bw} = \sum_{l=1}^{h}(k \times (k-1)^{l-1} \times S(l))$$

By substituting $k = d(G)$ and $S(l)$ from (4.14) into the above equation, b_{bw} can be written as

$$b_{bw} = \frac{b_{es} \times d(G) \times \left(1 + (h \times (d(G) - 2) - 1) \times (d(G) - 1)^h\right)}{(2 - d(G))^2}$$

Let b_{pa} be the size of an entity-tuple pair in bytes, and η be the average number of entity-tuple pairs of the essential-set which have the same alternative of Q as their first element. Then, b_{es}, i.e. the average size of the essential-set in each node, can be computed as $b_{es} = |Q| \times \eta \times b_{pa}$.

In Section 4.6, we show that η is very small and almost independent from the number of tuples which are maintained at a node. However, η is dependent to the correlation between the probability of the tuples and their similarity to Q's alternatives.

Let us show with an example that b_{bw} is not significant. Consider that 10,000 nodes receive Q (including the query originator), thus $|P_Q| = 10,000$. Assume that $d(G) = 4$. Thus, the height of the query-tree, i.e. h, is equal to 8. Our experiments show that η is about 2.2 when similarity and probability are not correlated. Consider Q has two alternative tuples. Since the actual data contents of the entity-tuple pair (t_i, t_j) is not transferred during the Merge-and-Backward phase, we set b_{pa} to 23, i.e. 1 bytes for i, 4 bytes for j, 6 bytes for the address of the node in which t_j is maintained, 4 bytes for the x-tuple to which t_j belongs, 4 bytes for the similarity score of t_i to t_j, and 4 bytes for the probability of t_j. As a result, b_{bw} is less than 10 megabytes for a distributed system that contains 10,000 nodes.

4.5.4 Retrieve Messages

By retrieve messages, we mean the message sent by the query originator to request the MPMP and the message sent by the node owning the MPMP to return it. Therefore, the number of retrieve messages, denoted by m_{rt}, is $m_{rt} = 2$. The total size of data transferred by these messages, denoted by b_{rt}, can be computed as $b_{rt} = m_{rt} \times b_t$, where b_t is the average size of a tuple.

4.6 Performance Evaluation

We evaluated the performance of FD through implementation and simulation. The implementation over a 75-node cluster was useful to validate our algorithm in a realistic experimental environment. The simulation allowed us to study the performance of our algorithm under various conditions.

The rest of this section is organized as follows. In section 4.6.1, we describe our experimental and simulation setup, and the algorithms used for comparison. In section 4.6.2, we evaluate the response time of FD. Section 4.6.3 presents the evaluation of communication cost based on the bandwidth usage and the number of exchanged messages among nodes. In Section 4.6.4, we present the result of applying FD on real data.

4.6.1 Experimental and Simulation Setup

In our implementation and simulation, we compare FD with two baseline algorithms. The first algorithm is a centralized algorithm which we call FC (Fully Centralized). With FC, all nodes that receive the query send their data to the query originator where the most probable matching pair is computed using a centralized algorithm. The details of the centralized processing by FC can be found in Chapter 3. The second comparing algorithm is denoted by SCC (Score Confidence Centralized). In SCC, every node q receiving uncertain entity e (as the query) extracts a list containing the information of all of its pairs, and sends the extracted list directly to the query originator for centralized processing. More precisely, the information that is put in the sent list for each entity-tuple pair $(t_i, t_j) \in e \times D_q$, is a vector (i, a, j, x, s, p) where i is the index of t_i in e, a is the address of node q which owns t_j, j is the index of tuple t_j in the database D_q maintained by q, x is the x-tuple to which t_j belongs, s is the similarity score between t_i and t_j, and p is the probability of tuple t_j.

We implemented FD, FC, and SCC in Java, and tested them using a cluster of 75 nodes connected by a 1-Gbps network. Each node of cluster has a dual-quad-core 2.4 GHz processor and 24 GB memory. We make each node act as a node in the distributed system described in Section 4.5.1. We determined the node neighbors using the topologies generated by the BRITE universal topology generator [2]. Thus, each node only is allowed to communicate with the nodes that are its neighbors in the topology generated by BRITE.

To study the scalability of FD far beyond 75 nodes and to play with various performance parameters, we implemented a simulator using the PeerSim simulation kernel [5] and the Java programming language. We use the event driven engine of PeerSim to be able to simulate the delay in sending messages and also the bandwidth of nodes. We assign a random delay, denoted by latency, to communication ports to simulate the delay for sending a message between two nodes in a real distributed system. Also, we assign an upstream and a downstream bandwidth to each node. To simulate a node, we use a PeerSim's node that performs all tasks that must be done by a node for executing FD, FC, and SCC algorithms. We implemented each of the three algorithms as a protocol in PeerSim. We used PeerSim's WireKOut topology generator that randomly selects k neighbors for each node in the network. We used undirected links between nodes and set k to 10.

The experimental and simulation parameters are listed in Table 4.1. Notice that *bandwidth* and *latency* parameters are used only in our simulation. Unless otherwise specified, we use the values in this table for our tests. Each node has a table $R(data, sim, p)$ in which attribute *data* is a random real number with normal distribution with a mean of 1 KB (Kilobytes) and a variance of 16 KB, *sim* is a random real number in the interval [0..1] with normal distribution with a mean of 0.5 and a variance of 0.04, p is a random real number in the interval (0..1]

Table 4.1: Parameters

Parameter	**Values**
data: tuple's data items size	Gaussian random, Mean = 1 KB, Variance = 16 KB
sim: similarity score	Gaussian random, Mean = 0.5, Variance = 0.04
p: probability	Gaussian random, Mean = 0.4, Variance = 0.04
N: number of tuples at each node	Uniform random integer in range [4500..5500]
d_x: x-tuple's average alternatives	3 for R table and 2 for the query
Cor: correlation between *sim* and p	0
Upstream bandwidth	Gaussian random, Mean = 56 Kbps, Variance = 32 Kbps
Downstream bandwidth	8 × Upstream bandwidth
Latency	Gaussian random, Mean = 200 ms, Variance = 100 ms
Number of nodes	10,000
TTL: Time To Live	100

with normal distribution with a mean of 0.4 and a variance of 0.04. Attribute *data* represents the data item that is returned back to the user as the result of the query and its value simulates the size of the data item. Attribute *sim* is used for computing the similarity between the tuple and the tuples in the query, and attribute p is the confidence value of the tuple. We introduce a number of parameters to control the characteristics of the R table. The number of tuples in R is denoted by N. The average number of alternatives that an x-tuple can have is denoted by d_x. The correlation between *sim* and p is denoted by *Cor*. To generate each tuple in R, we use a normal distribution for generating attribute *data*, and a bivariate normal distribution with a given correlation for generating *sim* and p attributes. We repeat this process to generate N different tuples. Then, we generate d as a uniform random integer in $[1, 2 \times d_x - 1]$, 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 generate the query needed for experiments, in the same way that we generated a valid x-tuple. As Table 4.1 shows, we use the following default values for N, d_x, and *Cor* unless otherwise specified. N is a random number, uniformly distributed over all nodes, which is greater than 4500 and less than 5500. We use the default value $d_x = 3$ for the database and $d_x = 2$ for the query, and we use the default value $Cor = 0$.

For our implementation, we generate the R table and the query in the same way as our simulation, except for the data item size which is no longer simulated by a real number but with an array containing data.

Unless otherwise specified, we use the following values for the other simulation parameters. The *upstream bandwidth* of nodes is a random number with normal distribution with a mean of 56 Kbps (Kilobits per second) and a variance of 32 Kbps. The *downstream bandwidth* of each node is set to a value equal to 8 times of its *upstream bandwidth*. The *latency* for sending messages between any two nodes is also a random number with normal distribution with a mean of 200 milliseconds and a variance of 100 milliseconds.

Running the simulator on a machine with 16 GB of memory, allows us to perform tests up to 10,000 nodes, after which the simulation data no longer fit in RAM and makes our tests difficult. This is quite sufficient for our tests. Therefore, the number of nodes of the system is set to be 10,000, unless otherwise specified.

In all of our tests, we set *TTL* to a high value, i.e. 100, to be sure that all nodes receive the query although the maximum hop-distance to other nodes from the query originator is much less than 100 with the topology that we use for our distributed system.

We repeat each simulation 10 times with the same query but with a different random number seed and average the outcomes.

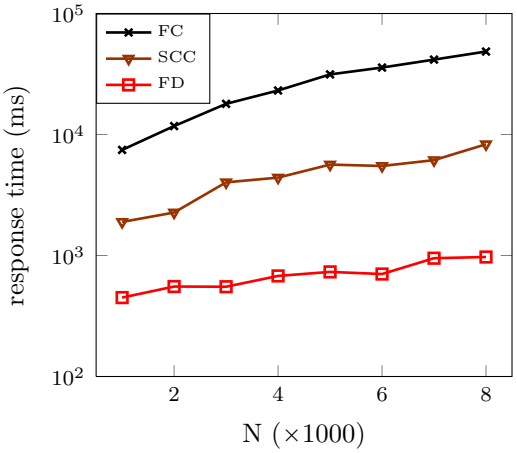


Figure 4.3: Response time vs. number of tuples (on cluster)

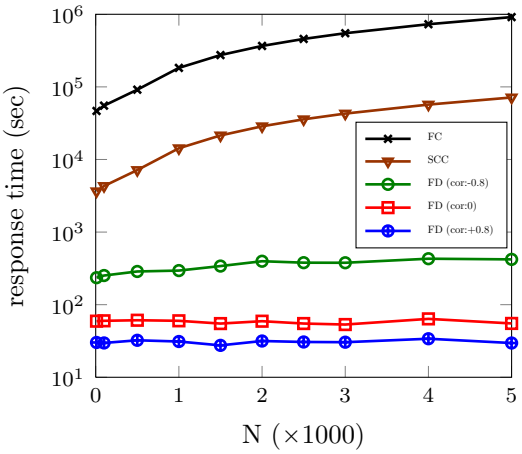


Figure 4.4: Response time vs. number of tuples

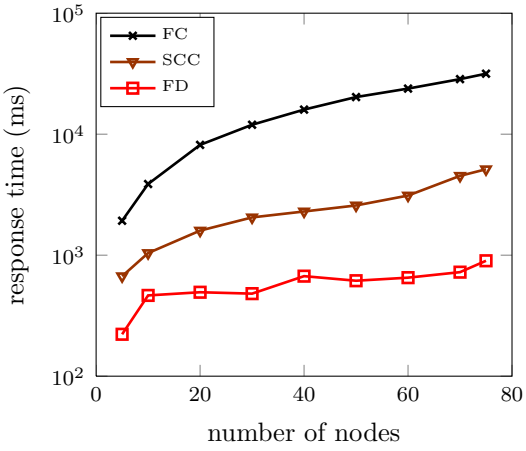


Figure 4.5: Response time vs. number of nodes (on cluster)

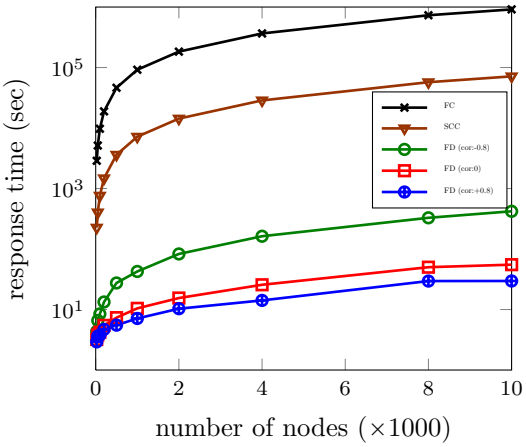


Figure 4.6: Response time versus number of nodes

4.6.2 Response Time

Scale up

In this section we study the response time of distributed entity resolution by varying the number of tuples, i.e. N, and the number of nodes. The response time is the time elapsed from submitting the query to a node to sending the result of the query to the user. The response time includes local processing time and data transfer time. To study the effect of different correlations between similarity and confidence values, we ran experiments using three different correlations, i.e. negative, zero, and positive correlations.

We used our implementation over the cluster to study how the response time increases with increasing the number of tuples in each node. Using implementation over the cluster, Figure 4.3 shows the response times of FD, FC, and SCC with N increasing up to 8,000. Using simulation, Figure 4.4 shows the response times of the three algorithms with N increasing up to 5,000 and the other simulation parameters set as in Table 4.1.

While FD significantly outperforms the other two algorithms, its response time is affected only very little with increasing N. As we expected, the negative correlation between similarity and confidence increases the response time since it increases the size of the essential-set at each node, and this increases the response time. Using independent random variables for similarity and confidence, i.e. zero correlation, decreases the response time and the positive correlation even decreases it more. Different correlations between similarity and confidence do not have any impact on FC and SCC algorithms, since they always send the whole database or the extracted similarity-confidence pairs of the whole database respectively.

We also used our implementation over the cluster to study the effect of the number of nodes on response time. Using implementation over the cluster, Figure 4.5 shows the response times of FD, FC, and SCC with the number of nodes increasing up to 75 and the other experimental parameters set as in Table 4.1. Using simulation, Figure 4.6 shows the response times of the three algorithms with the number of nodes increasing up to 10,000 and the other simulation parameters set as in Table 4.1. FD always significantly outperforms the other two algorithms and the performance difference increases significantly in the favor of FD as the number of nodes increases. These figures show excellent scale up of FD since response time logarithmically increases with increasing the number of nodes. We also observe that negative correlation between similarity and confidence increases the response time, but zero and positive correlations decrease the response time. Also, the performance difference between different correlations increases as the number of nodes increases.

The experimental results correspond with the simulation results. However, the response time of implementation over the cluster is better than that of simulation

because the cluster has a high-speed network.

To sum up, the reason of excellent scalability of FD versus both the database size and the number of nodes is its distributed execution. In FC and SCC algorithms, a central node, i.e. the query originator, is responsible for query execution, and this makes them inefficient.

Effect of Latency and Bandwidth

In this section, we study the effect of latency and bandwidth on response time. In the previous simulation tests the latency and upstream bandwidth were normally distributed random numbers with mean values of 200 ms and 56 Kbps respectively. In this test, we vary the mean values of the latency and bandwidth and study their effects on response time. For both experiments on bandwidth and latency, we set both N and the number of nodes to 5,000 and other simulation parameters set as in Table 4.1.

Figure 4.7 shows how response time decreases with increasing bandwidth. Increasing the bandwidth has strong, similar effect on all three algorithms. FD outperforms the other two algorithms for all tested bandwidths.

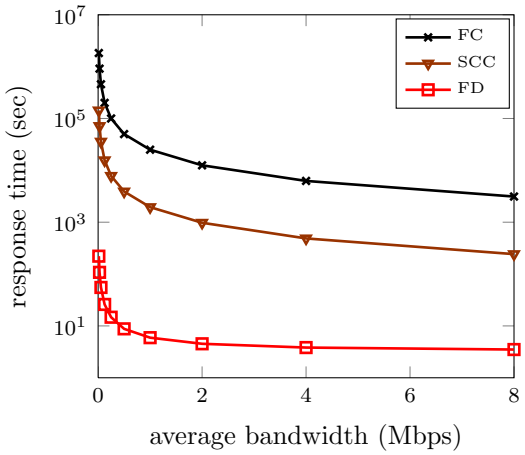


Figure 4.7: Effect of average bandwidth on response time

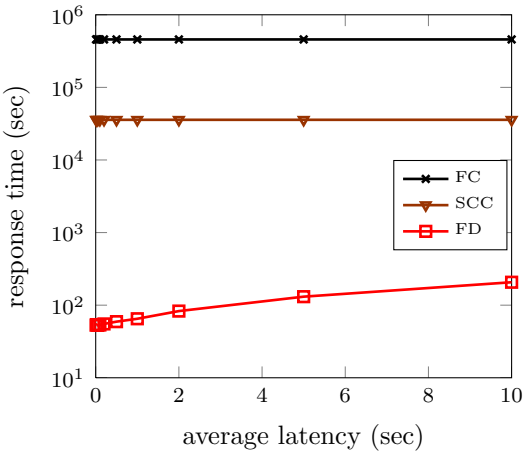


Figure 4.8: Effect of average latency on response time

Figure 4.8 shows how response time evolves with increasing latency. Latency has little effect on the FC and SCC algorithms, because in these algorithms the nodes return their results directly to the query originator, and do not bubble up the results. Although FD outperforms the other algorithms for all the tested values, high latency, e.g. more than 500 ms, has strong impact on it and increases its response time much. However, below 500 ms, latency does not have much effect on FD's response time.

4.6.3 Communication Cost

In this section, we study the communication cost of FD. We measure the communication cost in terms of the number of bytes, which should be transferred on the network for processing a query Q. We also measure the number of exchanged messages during the execution of an algorithms. To study the effect of different similarity-confidence correlations, we ran experiments using three different correlations, i.e. negative, zero, and positive correlations.

Figure 4.9 shows how communication cost evolves with the number of tuples in each node increasing up to 5,000 and the other simulation parameters set as in Table 4.1. This figure shows that FD significantly outperforms the other two algorithms. Moreover, while increasing N has strong effect on FC and SCC algorithms, it has a very little effect on FD. Also as in scalability experiments, negative correlation between similarity and confidence values increases the communication cost, and zero or positive correlation decreases it.

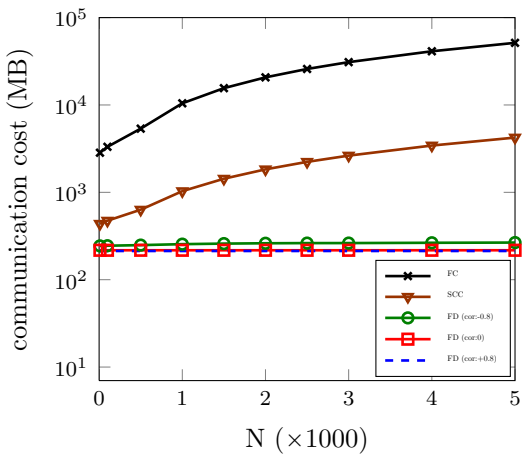


Figure 4.9: Effect of number of tuples on communication cost

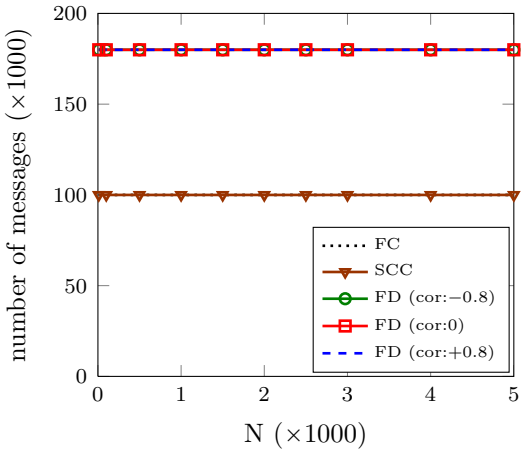


Figure 4.10: Number of exchanged messages vs. number of tuples

Figure 4.10 shows the number of messages exchanged during the execution of the three algorithms with the number of tuples in each node increasing up to 5,000 and the other simulation parameters set as in Table 4.1. This figure shows that the database size has no effect on the number of exchanged messages. Although FD exchanges more messages than the other two algorithms, since the sizes of these messages are much smaller than the sizes of the messages produced by the other two algorithms, FD's communication cost is significantly smaller than theirs.

Figure 4.10 also shows that different similarity-confidence correlations has no effect on the number of exchanged messages in FD.

We ran experiments to compare the average size of the essential-set with the number of entity-tuple pairs which exist at a node. To measure the average

essential-set size, we calculated the sum of the essential-set of all nodes and divided it by the number of nodes. In these experiments, we used uncertain entities with exactly 2 alternatives for the query. Figure 4.11 shows how the average size of the essential-set (in number of entity-tuple pairs) changes with the number of entity-tuple pairs in each node (i.e. $2 \times N$) increasing up to 10,000 and the other simulation parameters set as in Table 4.1. This Figure shows that the correlation between similarity and confidence has a strong effect on the size of the essential-set. The average size of the essential-set is almost constant for positive and zero correlations, i.e. 2 and 4.4 pairs respectively, but the essential-set size increases from 19.8 to 32.4 pairs for the negative correlation. These observations indicate that the size of the essential-set is very small and almost independent from the number of entity-tuple pairs which exist at the nodes. This means that our pruning algorithm performs quite effectively.

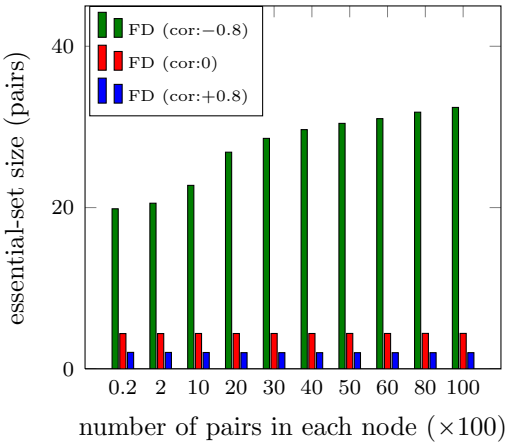


Figure 4.11: Essential-set size vs. number of pairs in each node

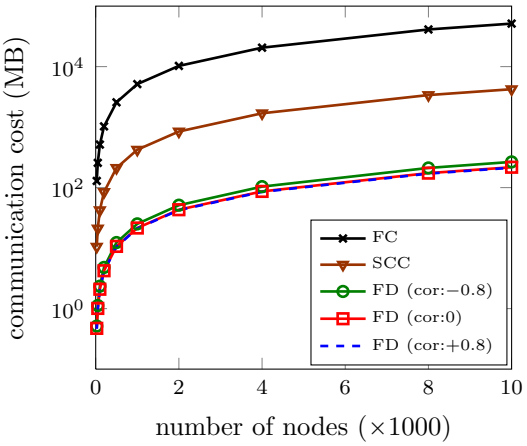


Figure 4.12: Effect of the number of nodes on the communication cost

We also ran experiments to study the effect of the number of nodes on communication cost. Figure 4.12 shows the communication costs of the three algorithms with the number of nodes increasing up to 10,000 and the other simulation parameters set as in Table 4.1. As this figure shows, FD significantly outperforms the other two algorithms and the performance difference increases significantly in the favor of FD as the number of nodes increases. Again as we expect, negative correlation between similarity and confidence increases the communication cost, but zero and positive correlations decrease the communication cost.

Figure 4.13 shows the number of messages exchanged during the execution of the three algorithms with the number of nodes increasing up to 10,000 and the other simulation parameters set as in Table 4.1. This figure shows that increasing the number of nodes increases the number of messages in the three algorithms. The number of exchanged messages in FD is higher than the other two algorithms but, as we discussed earlier because of the small size of these messages,

FD significantly outperforms the other algorithms based on communication cost. Again as we expect, different similarity-confidence correlations has no effect on the number of exchanged messages in FD.

4.6.4 Case Study on Real Data

In this section, we report the result of applying the three algorithms on real data.

As real-world database, we used a facial image database which we extracted from video. We downloaded 900 videos tagged with the keyword *"wedding ceremony"* from YouTube[3], and used 2 fps sampling method and the pittpatt software [6] to extract 5010 distinct facial images each associated with a confidence value, from the videos. Then, we used SIFT method [89] and the *bag of words* model [87] with a codebook of 250 visual words to represent each facial image with a vector containing 250 real numbers in range [0..1] each associated with a confidence value. We used the cosine similarity metric for measuring the similarity between vectors.

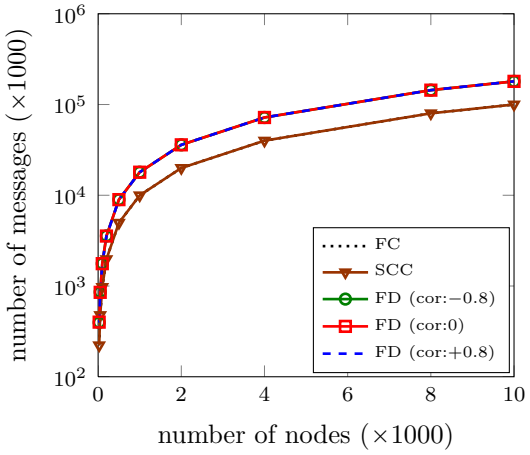


Figure 4.13: Effect of number of nodes on the number of exchanged messages

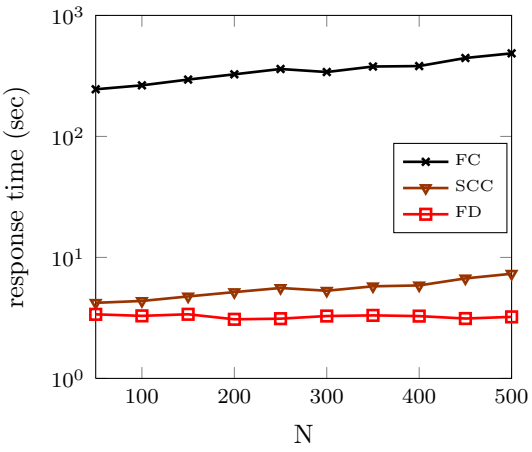


Figure 4.14: Response time vs. number of tuples (real data)

We set the number of nodes to 10; k to 5; and the other network parameters as in Table 4.1. In each experiment, we randomly selected one of the vectors as the query, and equally distributed N randomly selected vectors among the nodes in the network.

Figures 4.14 and 4.15, respectively, show how response time and communication cost increase with N increasing up to 5,000. With increasing N up to 5,000, Figure 4.16 compares the average size of the essential-set with the total number of pairs in each node. As we expected, the result of applying the algorithms on real data confirms the result we observe on synthetic data.

[3]http://www.youtube.com

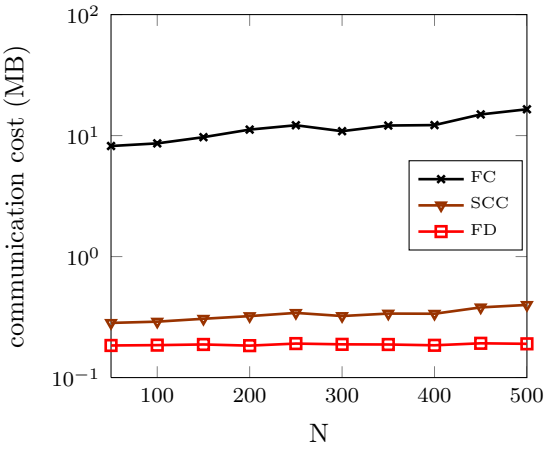


Figure 4.15: Communication cost vs. number of tuples (real data)

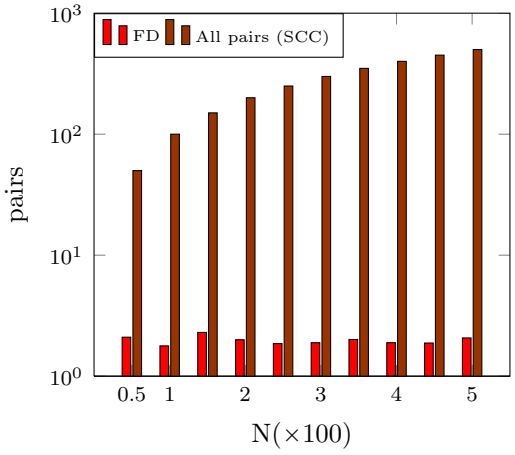


Figure 4.16: Essential-set size and all pairs in each node (real data)

4.7 Analysis against related Work

Beyond the work on entity resolution presented in Chapter 2, nearest neighbor and top-k queries over distributed uncertain data are also relevant to our research.

To the best of our knowledge, the existing nearest neighbor proposals, e.g. [44, 82, 125, 143], need the uncertain data to be stored in a centralized database, and thus cannot deal with the problem in distributed setting.

Recently, there have been some proposals dealing with the problem of top-k query processing for distributed uncertain data [140, 86]. In [140], the authors present a top-k query processing system for a wireless sensor network in which sensor nodes are grouped into clusters, where cluster heads are selected to perform localized data processing and to report aggregated results to the base station. Cluster heads use a user-specified probability threshold to find a rank boundary for pruning data gathered from sensors, before reporting to the base station. In [86], the authors present a proposal for ranking queries for distributed uncertain data. They use the concept of expected score and approximate it to reduce the communication cost and also processing time.

Our work nevertheless differs from these two above proposals because our problem definition is completely different. We look for an entity-tuple pair with the maximum probability of being the most similar pair, while [140] looks for tuples which have a probability higher than a user-specified threshold to be in the query result, and [86] is a proposal for approximating the expected score of the query results and then ranking them.

4.8 Conclusion

In this chapter, we proposed FD, a decentralized algorithm for dealing with the entity resolution problem over distributed probabilistic data, with the goal of minimizing the bandwidth usage. FD uses the novel concepts of *potential* and *essential-set* to prune data at local nodes. This leads to a significant reduction in bandwidth usage and response time compared to the baseline approaches. FD requires no global information, and does not depend on the existence of certain nodes.

We validated the performance of FD through implementation over a 75-node cluster and simulation using a simulator which we implemented using the PeerSim simulation kernel and the Java programming language. The experimental and simulation results show that response time of FD increases logarithmically with increasing the number of nodes. The experiments and simulations also show that FD's response time is almost independent from the size of the database in nodes. The results also show the excellent performance of FD, in terms of communication cost, compared with the two baseline algorithms, i.e. FC, and SCC.

Chapter 5

Entity Resolution for Probabilistic Data Using Entropy Reduction[1]

5.1 Introduction

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

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

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

[1]The material of this chapter has been partially published in [20].

more information about the criminals and can greatly speedup the investigation process.

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

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

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

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

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

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

5.2 Problem Definition

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

5.2.1 Data Model

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

We denote an uncertain database by $\mathcal{D}$, the set of its possible worlds by $PW(\mathcal{D})$, and the set of all tuples in $\mathcal{D}$ by D.

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

5.2.2 Entropy in Probabilistic Databases

In information theory, entropy is a measure of the uncertainty associated with a random variable [119]. It is usually used for quantifying the expected uncertainty (or quality) of communicated information. In this chapter, we use entropy to

x-tuple	t	$p(t)$
x_1	t_1	0.6
	t_2	0.3
x_2	t_3	0.2

(a)

w	w members	$p(w)$
w_1	$\varnothing$	$(1-p(t_1)-p(t_2)).(1-p(t_3)) = 0.08$
w_2	$\{t_1\}$	$p(t_1).(1-p(t_3)) = 0.48$
w_3	$\{t_2\}$	$p(t_2).(1-p(t_3)) = 0.24$
w_4	$\{t_3\}$	$(1-p(t_1)-p(t_2)).p(t_3) = 0.02$
w_5	$\{t_1, t_3\}$	$p(t_1).p(t_3) = 0.12$
w_6	$\{t_2, t_3\}$	$p(t_2).p(t_3) = 0.06$

(b)

Figure 5.1: a) An example probabilistic database $\mathcal{D}$ in the x-relation model, b) Possible worlds of $\mathcal{D}$

measure the expected uncertainty in a probabilistic database. For this, we consider the probabilistic database as a set of probabilistic deterministic database instances (i.e. the possible worlds of the probabilistic database). Below, we formally define the entropy of a probabilistic database.

5.2.1. Definition. *Entropy of a probabilistic database.* Let $\mathcal{D}$ be a probabilistic database, $PW(\mathcal{D})$ be the possible worlds of $\mathcal{D}$, and $p(w)$ be the probability of a possible world w. The entropy of $\mathcal{D}$ is defined as

$$entropy(\mathcal{D}) = -\sum_{w \in PW(\mathcal{D})} p(w).\log p(w). \tag{5.1}$$

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

5.2.3 Entity Resolution Over Probabilistic Database

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

However, the merging phase of ERPD is different from that of ER, because the merge function in ERPD should adjust the probabilities of the tuples to satisfy the constraints of the probabilistic data model. For instance, consider two non-resolvable conflicting[2] duplicate tuples t_1 and t_2 with probabilities 0.8 and 0.3 respectively, in the x-relation model. These tuples cannot be merged into a single

[2]See the definition in Section 2.2.

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

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

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

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

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

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

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

5.2.4 Problem Statement

In this chapter, we aim at merging the x-tuples while minimizing the entropy. Since we define the entropy as a measure of uncertainty in probabilistic databases,

we expect that the result of the merge phase be a database with minimum uncertainty, thus with high quality.

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

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

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

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

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

5.3 ERPD Using Entropy

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

5.3.1 Computing Entropy in X-relations

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

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

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

$$entropy(\mathcal{D}) = -\sum_{x \in \mathcal{D}} \sum_{t \in x} p(t).\log p(t) \tag{5.2}$$

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

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

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

$$entropy(\mathcal{D}') = -\sum_{w' \in PW(\mathcal{D}')} p(w').\log p(w') \tag{5.3}$$

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

$$PW(\mathcal{D}') = \{w' \mid w' = w \cup \{t\}, w \in PW(\mathcal{D}), t \in x_{k+1}\} \tag{5.4}$$

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

$$\begin{aligned}
&entropy(\mathcal{D}') = -\sum_{t \in x_{k+1}} \sum_{w \in PW(\mathcal{D})} (p(t).p(w).\log(p(t).p(w))) = \\
&-\sum_{t \in x_{k+1}} \sum_{w \in PW(\mathcal{D})} p(t).p(w).\log p(t) + \sum_{t \in x_{k+1}} \sum_{w \in PW(\mathcal{D})} p(t).p(w).\log p(w) = \\
&-\sum_{t \in x_{k+1}} p(t).\log p(t).\sum_{w \in PW(\mathcal{D})} p(w) + \sum_{w \in PW(\mathcal{D})} \sum_{t \in x_{k+1}} p(t).p(w).\log p(w) = \\
&-\sum_{t \in x_{k+1}} p(t).\log p(t).\sum_{w \in PW(\mathcal{D})} p(w) + \sum_{w \in PW(\mathcal{D})} p(w).\log p(w).\sum_{t \in x_{k+1}} p(t)
\end{aligned} \tag{5.5}$$

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

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

Using induction assumption, we have

$$entropy(\mathcal{D}') = -\sum_{t \in x_{k+1}} p(t).\log p(t) + \sum_{x \in \mathcal{D}} \sum_{t \in x} p(t).\log p(t) =$$
$$-\sum_{x \in \mathcal{D}'} \sum_{t \in x} p(t).\log p(t) = RHS(5.2).$$

□

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

5.3.2 Merge Function

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

Merge of Tuples

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

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

- **Mergeable type 1:** the merge result is not equal to any of the two tuples.
- **Mergeable type 2:** the merge result is only equal to one of the two tuples.
- **Mergeable type 3:** the merge result is equal to both tuples.

Notice that $t_\perp$ is mergeable with every tuple and the result of the merge is equal to that tuple. In other words, $t_\perp$ is the neutral element of $\odot$ operation.

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

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

CAF Merge Function

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

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

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

2. Normalizing the tuples' probabilities by a normalization factor k.

Any merge function that proceeds the above steps is commutative, but not necessarily associative. In fact, the value of the normalization factor greatly affects the associativity of the merge function. In CAF, we set the normalization factor equal to $1-p_\varnothing$, where $p_\varnothing$ is the aggregated probability of the contradictory possible worlds[3], i.e. W_c. As shown in [79], the only normalization factor for which the merge function is associative is $1 - p_\varnothing$. Indeed, this normalization factor normalizes the tuples' probabilities so that their sum is equal to one. Let us now formally define the CAF merge function.

[3]This value is equal to the normalization factor of Dempster's rule of combination [118].

5.3.3. Definition. **CAF merge function.** Let x and x' be two x-tuples. Let $p_\varnothing = \sum_{t \odot t' = \varnothing} p(t).p(t')$, where $t \in x$ and $t' \in x'$. Then, the result of merging x and x' by CAF is an x-tuple as follows:

$$x \odot x' = \{t'', p(t'') = \frac{\sum_{t \odot t' = t''} p(t).p(t')}{1 - p_\varnothing} \mid t \in x, t' \in x', t'' \neq \varnothing\}$$

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

x-tuple	t	name	phone	address	p(t)
x_1	t_1	Thomas Michaelis	5256661	21 College Blvd.	0.4
	t_2	Thomas Michaelis	$\perp$	45, Main street	0.5
x_2	t_3	T. Michaelis	5256661	$\perp$	0.3
	t_4	T. Michaelis	5256661	45, Main street	0.2
	t_5	Thomas Michaelis	5256661	45, Main st.	0.2

(a)

$\odot$	t_1	t_2	$t_\perp$
t_3	$t_1 \odot t_3 = \varnothing$, 0.12	$t_2 \odot t_3 = t'$, 0.15	$t_\perp \odot t_3 = t_3$, 0.03
t_4	$t_1 \odot t_4 = \varnothing$, 0.08	$t_2 \odot t_4 = t'$, 0.1	$t_\perp \odot t_4 = t_4 = t'$, 0.02
t_5	$t_1 \odot t_5 = \varnothing$, 0.08	$t_2 \odot t_5 = t'$, 0.1	$t_\perp \odot t_5 = t_5 = t'$, 0.02
$t_\perp$	$t_1 \odot t_\perp = t_1$, 0.12	$t_2 \odot t_\perp = t_2$, 0.15	$t_\perp \odot t_\perp = t_\perp$, 0.03

(b)

x-tuple	t	p(t)
$x_1 \odot x_2$	t_1	0.12/0.72 = 0.17
	t_2	0.15/0.72 = 0.21
	t_3	0.03/0.72 = 0.04
	t'	(0.15 + 0.1 + 0.1 + 0.02 + 0.02)/0.72 = 0.54

(c)

Figure 5.2: a) Example of two x-tuples and their alternatives, b) CAF's approach in merging the two x-tuples, c) The result of merging by CAF

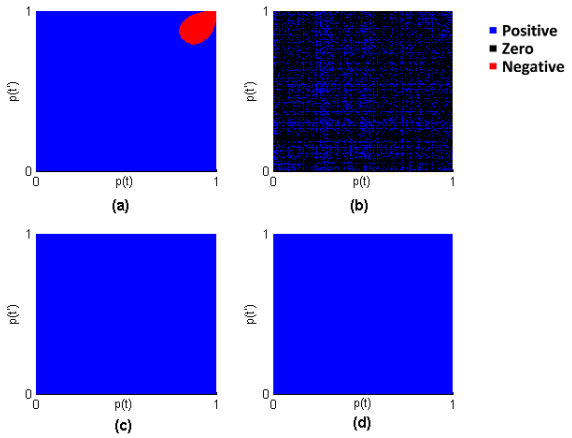


Figure 5.3: Entity reduction for two x-tuples $x = \{t\}$ and $x' = \{t'\}$ when t and t' are: a) Non-mergeable, b) Mergeable type 1, c) Mergeable type 2, d) Mergeable type 3. Blue, black, and red colors represent entropy reductions with positive, zero, and negative values respectively.

Entropy Reduction Property

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

$$\textit{entropy-reduction-value} = \textit{entropy}(x) + \textit{entropy}(x') - \textit{entropy}(x \odot x') > 0 \tag{5.6}$$

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

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

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

5.3.3 ME Algorithm

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

Given a single-alternative x-relation $\mathcal{D}$, ME works as follows:

1. Ignoring the tuples' probabilities, use an existing duplicate detection method to partition D into a set E of partitions such that each partition contains duplicate tuples which are believed to represent the same real-world entity.
2. For each partition E_i in E do steps 3 to 6.
3. Build the set X_i of x-tuples by putting each tuple of E_i in an x-tuple.
4. Let x and x' be two x-tuples in set X_i for which the entity reduction value is maximum, say r_{max}.
5. If $r_{max} > 0$, then remove x and x' from X_i; add $x \odot x'$ to X_i; and goto step 4.
6. Add all of the x-tuples of X_i to database $\mathcal{D}^c$.
7. Return $\mathcal{D}^c$.

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

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

Let us now illustrate ME using an example.

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

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

t	A	p(t)
t_1	a	0.5
t_2	a	0.9
t_3	b	0.8
t_4	c	0.7
t_5	c	0.6
t_6	d	0.1
t_7	e	0.3

(a) Database $\mathcal{D}$

x-tuple	t	A	p(t)
x_1	$t_{1,2}$	a	0.79
	t_3	b	0.16
x_2	$t_{4,5}$	c	0.88
x_3	t_6	d	0.07
	t_7	e	0.28

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

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

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

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

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

5.3.4 Time Complexity

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

Let us analyze the time that ME spends for each partition. Let d be the number of tuples in the partition. In the first stage, ME computes the entropy reduction value for all $\frac{1}{2}d(d-1)$ single-alternative x-tuple pairs in the partition and

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

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

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

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

5.3.5 Multi-Alternative X-relation Case

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

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

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

3. Return E

5.4 Performance Evaluation

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

5.4.1 Experimental Setup

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

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

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

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

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

Table 5.1: Real-world databases used in our tests

name	attributes	tuples	entities
Cora	12	1295	117
Restaurant	4	864	751

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

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

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

Table 5.2: Default setting of experimental parameters

Parameter	**Values**
n: number of tuples in the database	200,000
d_e: average number of duplicate tuples per entity	10
m_p: percentage of mergeable duplicate tuples	20%

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

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

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

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

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

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

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

the *query-result-error* as

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

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

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

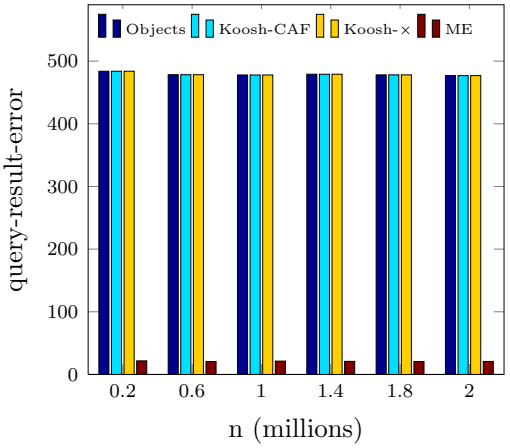


Figure 5.5: Query-result-error vs. n

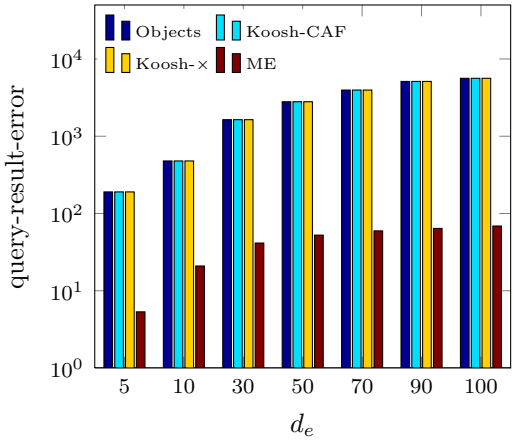


Figure 5.6: Query-result-error vs. d_e

5.4.2 Performance Results

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

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

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

Evaluating the Quality of the Query Results

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

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

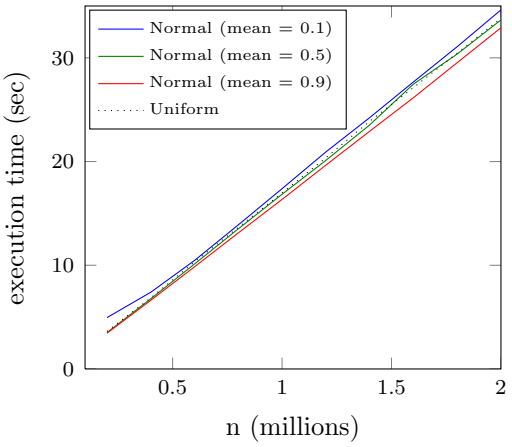


Figure 5.7: Execution time of ME vs. n over normal and uniform databases

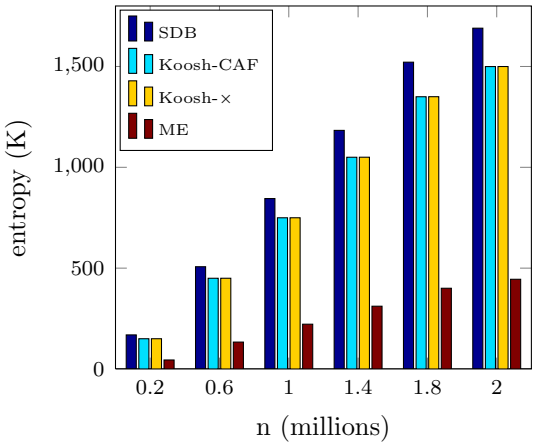


Figure 5.8: Entropy vs. n over normal database with $mean = 0.5$

Figure 5.6 shows how the query-result-error in the query results over the *Objects*, Koosh-CAF, Koosh-×, and ME databases increases with d_e increasing up to 100, n set to 200,000, and m_p set to 0. For the same reason as above, we observe that ME drastically reduces the query-result-error in the query results,

while Koosh-CAF and Koosh-× do not change it. Figures 5.5 and 5.6 also show that in ME, while the reduction in query-result-error is almost constant with n, it is linear with d_e. The reason is that the error in the result of a COUNT query over a probabilistic database is directly affected by the sum of the probabilities of the duplicate tuples in each entity, which is linear with d_e; as a result, increasing d_e has a linear effect on the query-result-error over the *Objects* database. In the ME database, we combine all duplicate tuples of an entity to a number of x-tuples. In each x-tuple, the sum of probabilities of the tuples is less than 1 and the number of x-tuples in the ME database is almost independent from d_e, thus query-result-error over the ME database is almost independent from d_e. Thus, the reduction in query-result-error is linear in d_e.

Effect of the Number of Tuples

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

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

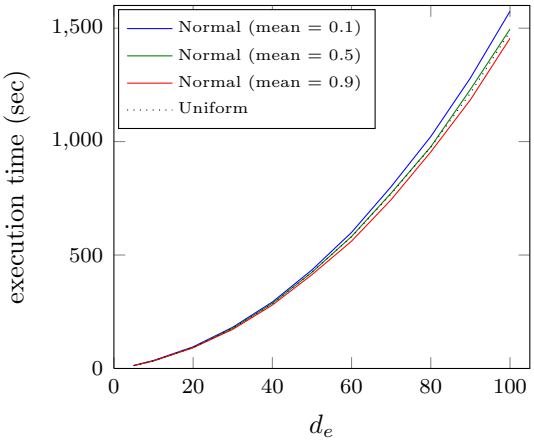


Figure 5.9: Execution time of ME vs. d_e over normal and uniform databases

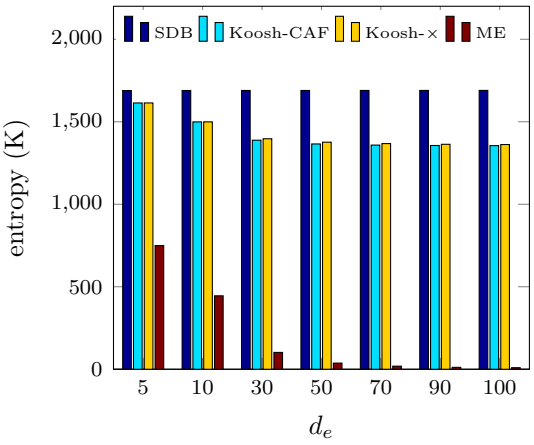


Figure 5.10: Entropy vs. d_e over normal database with $mean = 0.5$

With the number of tuples increasing up to 2,000,000 and the other parameters set as in Table 5.2, Figure 5.8 compares the entropy of the SDB database with that of the databases produced by Koosh-CAF, Koosh-×, and ME algorithms for normal SDB database with $mean = 0.5$. The other tested normal and uniform

distributions also show the same trend. This figure shows that while entropy reduction in Koosh-CAF and Koosh-× is not significant, ME significantly reduces entropy, i.e. about 73%. We also observe that the entropy of the databases are linear in n in all distributions. However, different distributions do affect the coefficient in the linear relation between the *entropy* and n: a very low or high mean value for p (e.g. 0.1 and 0.9) decreases it, but a value close to 0.5 increases it.

Effect of the Average Number of Duplicate Tuples per Entity

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

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

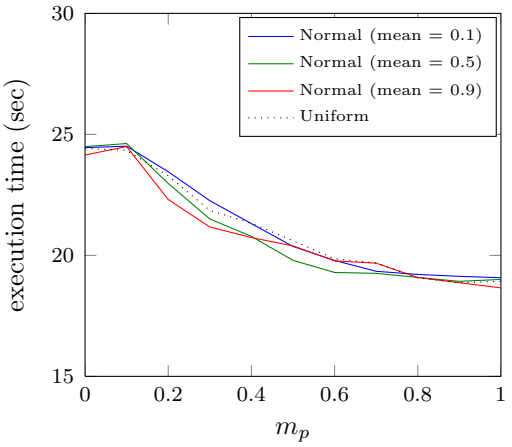


Figure 5.11: Execution time of ME vs. m_p over different databases

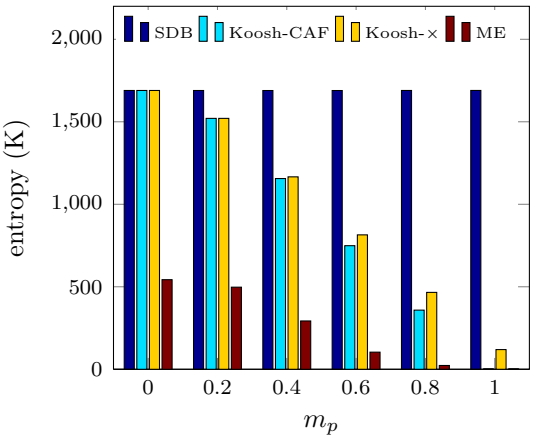


Figure 5.12: Entropy vs. m_p over normal database with $mean = 0.5$

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

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

Effect of the Percentage of Mergeable Duplicate Tuples

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

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

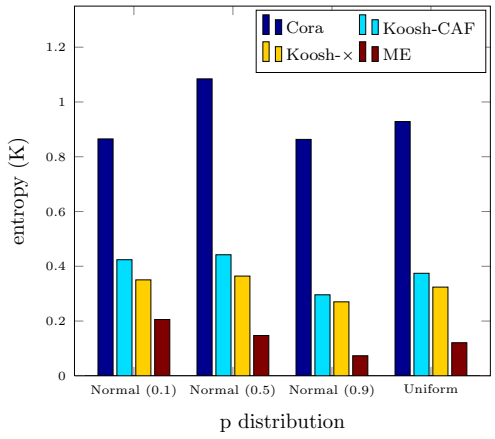


Figure 5.13: Result of applying the algorithms to the Cora database

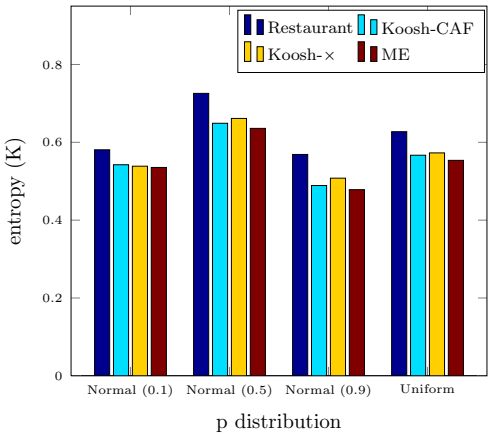


Figure 5.14: Result of applying the algorithms to the Restaurant database

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

distributions. Again, ME outperforms the other algorithms. These figures show that, while the entropy of the SDB database remains constant, the entropy of the other databases decreases significantly with increasing m_p. For instance, we observe that ME achieves 99% reduction in entropy over the normal SDB database with $mean = 0.5$ and $m_p = 1$. The reason for this observation in Koosh-CAF and ME is that when m_p increases, more tuples are merged together. In Koosh-×, the resulted merged tuples have lower probability than the original tuples, and this results in discarding the merged tuples. Hence, the entropy of Koosh-× is always higher than that of Koosh-CAF and ME.

Results on Real Data

Let us report the results of evaluating the three algorithms over the two real-world databases, i.e. Cora and Restaurant. We use real data in order to cover all practical cases which happen in real-world, particularly the merge types that we do not cover in the synthetic data, i.e. merge types 1 and 2 (refer to Section 5.3.2 for definition).

Figure 5.13 compares the entropy of the Cora database with the entropy of the databases produced by Koosh-CAF, Koosh-×, and ME algorithms over different distributions of p. We observe that ME signifcantly outperforms the other algorithms. This figure shows that ME significantly reduces the entropy of the Cora database over all p distributions. For instance, we observe about 91% reduction in entropy over the normal database with $mean = 0.9$.

Figure 5.14 shows the entropy of the Restaurant database with the entropy of the databases produced by Koosh-CAF, Koosh-×, and ME algorithms over different distributions of p. This figure shows that all three algorithms equally reduce the entropy of the Restaurant database over all p distributions. However, the reduction is much less than that of the Cora database. The reason is that Cora and Restaurant databases have different number of duplicate tuples per entity (i.e. on average 11.1 for Cora database and 1.2 for Restaurant database), and this has a direct effect on the entropy reduction.

5.5 Analysis against related Work

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

In the database literature, entropy has been used as a quality metric for query results over a probabilistic database [43, 42]. In [43], entropy is used to measure the quality of the range queries, nearest neighbor queries, AVG and SUM queries over a probabilistic database which contains a probabilistic attribute, modeled using a probability density function over an interval. In [42], the authors define entropy as a quality metric for the query results over a probabilistic database, and

propose a data cleaning algorithm with the goal of optimizing the expected quality improvement of the query results under a limited budget. Although entropy is meant to act as a quality metric for any query, the proposal in [42] can efficiently approximate it only for a particular subclass of queries over an x-relation. Our work differs from these proposals however in the way that we use entropy as a quality metric for the probabilistic database, rather than only for query results over it, and we propose an efficient approach for entity resolution using entropy.

5.6 Conclusion

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

Chapter 6

Pay-As-You-Go Data Integration Using Functional Dependencies[1]

6.1 Introduction

In most of the applications that the entity resolution (ER) problem arises, the data resides in a number of data sources with heterogenous schemas. Thus, before dealing with the ER problem, we first have to deal with the schema heterogeneity problem of data sources. Our aim in this chapter is to deal with this problem by setting up a data integration system without human intervention.

Data integration systems offer uniform access to a set of autonomous and heterogeneous data sources. Sources may range from database tables to web sites, and their numbers can range from tens to thousands. The main building blocks of a typical data integration application are the mediated schema definition, schema matching and schema mapping. The mediated schema is the integrated schema on which users pose queries. Schema matching is the process of finding associations between the elements (often attributes or relations) of different schemas, e.g. one source schema and the mediated schema in the popular Local As View (LAV) approach [100]. Schema mapping (also referred to as semantic mapping) is the process of relating the attributes of source schemas to the mediated schema (sometimes using expressions in a mapping language). The output of schema matching is used as input to schema mapping algorithms [100, 130].

Setting up a full data integration system with a manually designed mediated schema requires significant human effort (e.g. by domain experts and database designers). On the other hand, the applications that we consider in this thesis need to start with a data integration application in a complete automatic setting for reducing human effort and development time, and putting more effort into improving it, as needed. This setting is referred to by pay-as-you-go data integration.

[1]The material of this chapter has been partially published in [19] and [18].

The goal of our research is to study how advanced of a starting point can we build *a pay-as-you-go data integration system* in a fully automated setting. Since probabilistic data models have shown to be promising [59, 115], we build our approach on a probabilistic data model to capture the uncertainty that arises during the schema matching process. Therefore, we generate a set of Probabilistic Mediated Schemas (PMSs). The idea behind PMSs is to have several mediated schemas, each one with a probability that indicates the closeness of the corresponding mediated schema to the ideal mediated schema.

In database literature, the closest related work to ours is that of Sarma et al. [115] which is based on the PMSs proposed UDI (Uncertain Data Integration), as an uncertain data integration system. However, UDI may fail to capture some important attribute correlations, and thereby produce low quality answers. Let us clarify this by an example, which is the same as the running example in [115].

6.1.1. Example. Consider the following schemas both describing people:
$S_1(name, hPhone, hAddr, oPhone, oAddr)$
$S_2(name, phone, address)$
In S_2, the attribute *phone* can either be a home phone number or an office phone number, and the attribute *address* can either be a home address or an office address.

A high quality data integration system should capture the correlation between *hPhone* and *hAddr* and also between *oPhone* and *oAddr*. Specifically, it must generate schemas which group the *address* and *hAddr* together if *phone* and *hPhone* are grouped together. Similarly it should group the *address* and *oAddr* together if *phone* and *oPhone* are grouped together. In other words both of the following schemas should be generated (we abbreviate *hPbone*, *oPhone*, *hAddr*, *oAddr* as hP, oP, hA, and oA respectively):
$M_1(\{name, name\}, \{phone, hP\}, \{oP\}, \{address, hA\}, \{oA\})$
$M_2(\{name, name\}, \{phone, oP\}, \{hP\}, \{address, oA\}, \{hA\})$

UDI does not consider attribute correlations. Thus, UDI may generate M_1 and M_2 together with many other schemas that do not always respect attribute correlations. As a results, by producing a large number of schemas which can easily be exponential, the desirable schemas get a very low probability.

Most attribute correlations are expressed through Functional Dependencies (FDs), which are defined among them. For example let F_1 and F_2 be the set of FDs of S_1 and S_2 respectively:
$F_1 = \{hPhone \rightarrow hAddr, oPhone \rightarrow oAddr\}$
$F_2 = \{phone \rightarrow address\}$
FDs in F_1 and F_2 show the correlation between the attributes in S_1 and S_2, respectively. For example, $hPhone \rightarrow hAddr$ indicates that the two attributes *hPhone* and *hAddr* are correlated. Considering the pairs of FDs from different

sources can help us with extracting these correlations and achieving the goal of generating mediated schemas that represent these correlations. For example, the FD pair: $phone \rightarrow address$ and $hPhone \rightarrow hAddr$ indicate that if we group *phone* and *hPhone* together, we should also group *address* and *hAddr* together, and similarly *oPhone* and *oAddr*.

In this chapter, we take advantage of the background knowledge which is implied within functional dependencies (FDs), for building a pay-as-you-go data integration system. The specific contributions of this chapter are the following.

- We propose IFD (Integration based on Functional Dependencies), a data integration system that takes into account attribute correlations by using functional dependencies, and captures uncertainty in mediated schemas using a probabilistic data model. Our system allows integrating a given set of data sources, as well as incrementally integrating additional sources, without needing to restart the process from scratch.
- We model the schema matching problem as a clustering problem with constraints. This allows us to generate mediated schemas using algorithms designed for the latter problem. In our approach, we build a custom distance function for representing the knowledge of attribute semantics which we extract from FDs.
- We propose a new metric (i.e. FD-point) for ranking the generated mediated schemas in the clustering process, and selecting high quality ones.
- To validate our approach, we have implemented it as well as the baseline solutions. The performance evaluation results show significant performance gains of our approach in terms of recall and precision compared to the baseline approaches. They confirm the importance of FDs in improving the quality of uncertain mediated schemas.

The rest of the chapter is organized as follows. In Section 6.2, we make our assumptions precise and define the problem. In Section 6.3, we briefly describe the architecture of our data integration system. In Section 6.4, we propose our approach for schema matching. We also analyze the execution cost of our proposed approach. Section 6.5 describes our performance validation. Section 6.6 discusses related work, and Section 6.7 concludes the chapter.

6.2 Problem Definition

In this section, we first give our assumptions and some background about PMSs. Then, we state the problem addressed in this chapter.

We assume that the functional dependencies between the attributes of sources are available. This is a reasonable assumption in the applications which we consider, because the data source providers are willing to provide the full database

design information, including functional dependencies. However, there are contexts, such as the web, for which functional dependencies are not available. For these applications, we can use one of the existing solutions, e.g. [77, 132], to derive functional dependencies from data. Second assumption, which we make for ease of presentation, is that the data model is relational.

Let us formally define some basic concepts, e.g. functional dependencies and mediated schemas, and then state the problem addressed in this chapter. Let S be a set of source schemas, say $S = \{S_1, \ldots, S_n\}$, where for each $S_i, i \in [1, n], S_i = \{a_{i,1}, \ldots, a_{i,l_i}\}$, such that $a_{i,1}, \ldots, a_{i,l_i}$ are the attributes of S_i. We denote the set of attributes in S_i by $att(S_i)$, and the set of all source attributes as A, i.e. $A = \cup_i att(S_i)$. For simplicity, we assume that S_i contains a single table. Let F be the set of functional dependencies of all source schemas, say $F = \{F_1, \ldots, F_n\}$. For each $S_i, i \in [1, n]$, let F_i be the set of functional dependencies among the attributes of S_i, i.e. $att(S_i)$, where each $fd_j, fd_j \in F_i$ is of the form $L_j \rightarrow R_j$ and $L_j \subseteq att(S_i), R_j \subseteq att(S_i)$. In every F_i, there is one fd of the form $L_p \rightarrow R_p$, where $R_p = att(S_i)$, i.e. L_p is the primary key of S_i.

We assume one-to-one mappings of source attributes, meaning that each attribute can be matched with at most one attribute. We do this for simplicity and also because this kind of mapping is more common in practice. For a set of sources S, we denote $M = \{A_1, \ldots, A_m\}$ as a mediated schema, where $A_i \subseteq A$, and for each $i, j \in [1, m], i \neq j \Rightarrow A_i \cap A_j = \emptyset$. Each attribute involved in A_i is called a mediated attribute. Every mediated attribute ideally consists of source attributes with the same semantics.

Let us formally define the concept of probabilistic mediated schemas (PMSs). The probabilistic mediated schemas (PMSs) for a set S of source schemas is the set $N = \{(M_1, P(M_1)), \ldots, (M_k, P(M_k))\}$ where

- M_i is a mediated schema for S, where $i \in [1, k]$.
- For each $i, j \in [1, k], i \neq j \Rightarrow M_i \neq M_j$, i.e. M_i, M_j are different clusterings of $att(S)$.
- $P(M_i) \in (0, 1]$.
- $\sum_{i=1}^{k} P(M_i) = 1$.

We use the precision, recall, and F-measure of the clustering for measuring the quality of the generated mediated schemas.

Now, we formally define the problem we address. Suppose we are given a set of source schemas S, and a set of functional dependencies F and a positive integer number k as input. Our problem is to efficiently find a set of k probabilistic mediated schemas that have the highest F-measure.

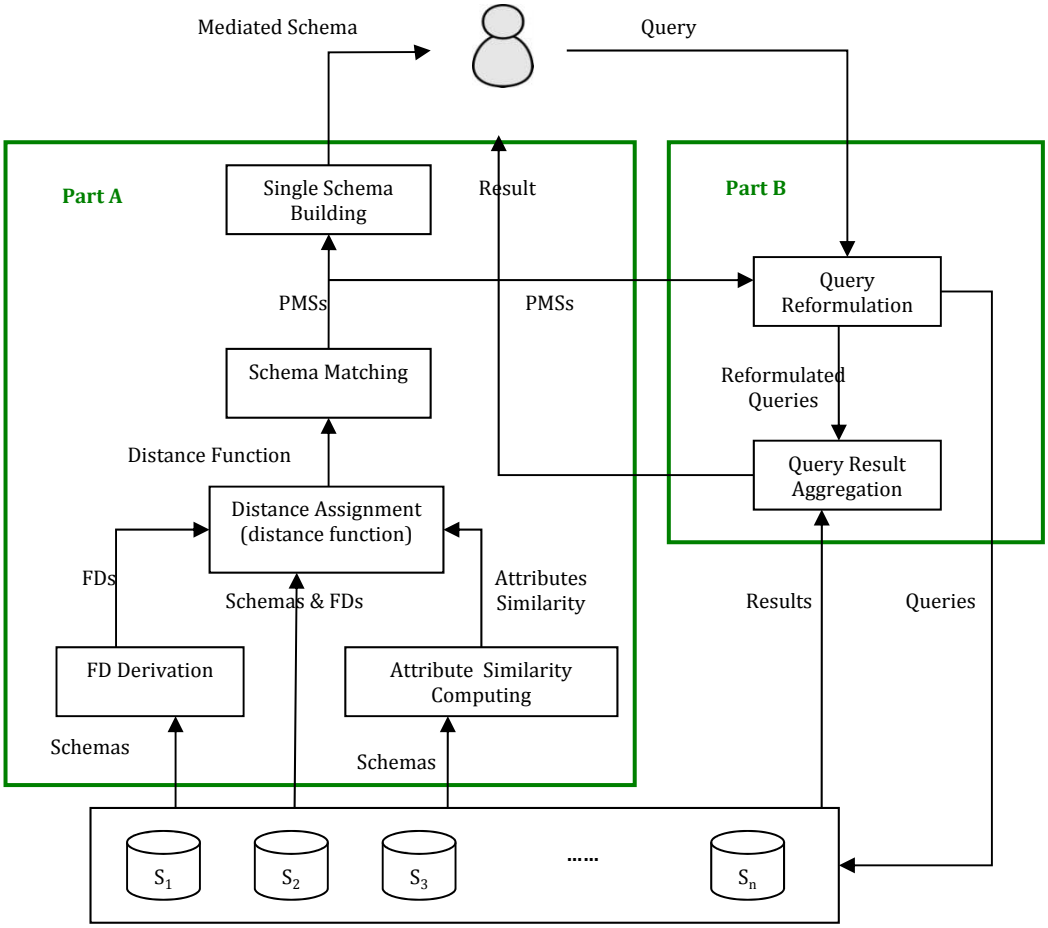


Figure 6.1: IFD architecture

6.3 System Architecture

The architecture of our data integration system, i.e. IFD, is shown in Figure 6.1. IFD consists of two main parts: schema matching (part A) and query processing (part B). The components of schema matching, which operate during the set-up time of the system, are as follows:

- *Attribute similarity computing*: this component computes the attribute name similarity between every two source attributes.

- *FD derivation*: this component derives functional dependencies from data, which is an optional component of the system and is only used in the cases where functional dependencies are not given to the system.

- *Distance assignment*: this component uses attribute pairs similarity and functional dependencies for generating the distance function.

- *Schema matching*: this component uses the distance function for generating a set of probabilistic mediated schemas.

- *Single schema building*: this component generates one mediated schema for the user by using the generated probabilistic mediated schemas.

The components of the query processing part is depicted in Part B of Figure 6.1. We include these components in the architecture of our system to provide a complete picture of a data integration system but our focus is on the schema matching part (part A). The components of part B which operate at query evaluation time are as follows:

- *Query reformulation*: This component uses the probabilistic mediated schemas to reformulate the user query posed against the mediated schema to a set of queries posed over the data sources. Our simplifying assumptions, i.e. one-to-one mappings and single-table data sources, greatly simplify the way this component reformulates the user query.

- *Query result aggregation*: This component combines the results of reformulated queries and assigns a probability to every tuple in the result, based on both the probabilities of the mediated schemas and the dependency among data sources.

6.4 Schema Matching

In this section, we present the schema matching part of IFD. To match the schemas automatically, we cluster the source attributes by putting semantically equivalent attributes in the same cluster. We use a clustering algorithm that works based on a *distance function*, which determines the distance between every two attributes. Specifically, we use the single-link CAHC (Constrained Agglomerative Hierarchical Clustering) algorithm [53]. In the rest of this section, we first describe our distance function. Then, we describe our approach for schema matching. We then describe a useful feature of our approach. Finally, we analyze the execution cost of the proposed algorithms.

6.4.1 Distance Function

Our schema matching algorithm uses a distance function for determining the distance between source attributes. To assign the distances between the attributes, we use the attributes' name similarity as well as some heuristics we introduce about FDs. In the rest of this section, we first describe our heuristics and then present the distance function algorithm.

FD Heuristics

We use heuristic rules related to FDs in order to assign the distance of attributes. Before describing our heuristics, let us first define *Match* and *Unmatch* concepts. Consider a_1 and a_2 as two typical attributes. If we want to increase their chance of being put in the same cluster, we set their distance to *MD* (i.e. Match Distance) which is 0 or a number very close to 0. In this case, we say that we matched a_1 with a_2, and we show this by $Match(a_1, a_2)$. In contrast, if we want to decrease their chance of being put in the same cluster, then we set their distance to *UMD* (i.e. Un-Match Distance) which is 1 or a number very close to 1. In this case, we say that we *unmatched* a_1 and a_2 and we show this by $Unmatch(a_1, a_2)$. Now, Let us use the following example to illustrate the heuristics.

6.4.1. Example. Consider two source schemas, both describing bibliographical information of scientific papers. In this example, primary keys are underlined; F_1 and F_2 are the sets of FDs of S_1 and S_2 respectively:

$S_1(\underline{author, title, year}, institution, journal, issn)$
$S_2(\underline{name, paper\_title, date}, affiliation, journal, serial\_no, volume, issue)$
$F_1 = \{author \rightarrow institution, journal \rightarrow issn\}$
$F_2 = \{name \rightarrow affiliation, journal \rightarrow serial\_no\}$

6.4.2. Heuristic. Let S_p and $S_q, p \neq q$, be two source schemas. Then,

$$Match(a_{p,i}, a_{q,k}) \Rightarrow unmatch(a_{p,i}, a_{q,l}) \wedge unmatch(a_{q,k}, a_{p,j})$$

where $a_{p,i} \in att(S_p), a_{p,j} \in att(S_p)\backslash\{a_{p,i}\}, a_{q,k} \in att(S_q), a_{q,l} \in att(S_q)\backslash\{a_{q,k}\}$.

Intuitively, this heuristic means that each attribute can be matched with at most one attribute of the other source.

6.4.3. Heuristic. Let $fd_p : a_{p,i} \rightarrow a_{p,j}$ and $fd_q : a_{q,k} \rightarrow a_{q,l}$ be two FDs, where $fd_p \in F_p, fd_q \in F_q, p \neq q$. Then, $similarity(a_{p,i}, a_{q,k}) > t_L \Rightarrow Match(a_{p,j}, a_{q,l})$ where t_L is a certain threshold and *similarity* is a given similarity function.

In this heuristic, We consider the set of facts that the two sources are assumed to be from the same domain, and both attributes $a_{p,j}$ and $a_{q,l}$ are functionally determined by the attributes $a_{p,i}$, and $a_{q,k}$ respectively, which themselves have close name similarity. Thus, we heuristically agree that: the probability of $Match(a_{p,j}, a_{q,l})$ is higher than that of $Match(a_{p,j}, a_{q,s})$ and $Match(a_{q,l}, a_{p,r})$, where $a_{q,s} \in att(S_q)\backslash\{a_{q,l}\}$ and $a_{p,r} \in S_p\backslash\{a_{p,j}\}$. Therefore, in such a case we match $a_{p,j}$ with $a_{q,l}$ to reflect this fact. Note that this heuristic has a general form in which there are more than one attribute on the sides of the FDs (see Section 6.4.1).

Let us apply heuristic 6.4.3 on Example 6.4.1. We have the FD $journal \rightarrow issn$ from S_1, and $journal \rightarrow serial\_no$ from S_2. There is only one attribute

at the left side of these FDs, and their name similarity is equal to 1 that is the maximum similarity value. Thus, we match the *issn* with the *serial_no* which appear on the right side of these FDs. Notice that approaches which only rely on name similarity, probably match *issn* with *issue*, which is a wrong decision.

6.4.4. HEURISTIC. Let PK_p and $PK_q, p \neq q$, be the primary keys of S_p and S_q respectively. Then,

$$(\exists a_{p,i} \in PK_p, a_{q,j} \in PK_q \mid (a_{p,i}, a_{q,j}) = \underset{a_p \in PK_p, a_q \in PK_q}{\arg\max} \; similarity(a_p, a_q)) \wedge$$
$$(similarity(a_{p,i}, a_{q,j}) > t_{PK}) \Rightarrow Match(a_{p,i}, a_{q,j})$$

where t_{PK} is a certain threshold and *similarity* is a given similarity function.

Let us explain the idea behind heuristic 6.4.4. Since we assume sources are from the same domain, there are a number of specific attributes which can be part of the primary key. Although these attributes may have different names in different sources, it is reasonable to expect that some of these attributes from different sources can be matched together. Obviously, we can set t_{PK} to a value less than the value we set for t_L because typically the probability of finding matching attributes in the primary key attributes is higher than the other attributes. After matching $a_{p,i}$ with $a_{q,j}$, we remove them from PK_p and PK_q respectively, and continue this process until the similarity of the pair with the maximum similarity is less than the threshold t_{PK} or one of the PK_p or PK_q has no more attributes to match.

Coming back to Example 6.4.1, it is reasonable to match the attributes: *author*, *title*, and *year* of S_1 with *name*, *paper_title*, and *date* of S_2 rather than with other attributes of S_2, and vice versa. The attribute pair with the maximum similarity is (*title*, *paper_title*). If we choose a good threshold, we can match these attributes together. The similarity of other attribute pairs is not high enough to pass the wisely selected threshold values.

6.4.5. HEURISTIC. Let PK_p and $PK_q, p \neq q$, be the primary keys of S_p and S_q respectively. Then,

$$(\exists a_{p,i} \in PK_p, a_{q,j} \in PK_q, fd_p \in F_p, fd_q \in F_q \mid$$
$$fd_p : a_{p,i} \rightarrow R_p, fd_q : a_{q,j} \rightarrow R_q) \Rightarrow Match(a_{p,i}, a_{q,j}) \qquad (6.1)$$

6.4.6. HEURISTIC. $(RHS(6.1) \wedge R_p = \{a_{p,r}\} \wedge R_q = \{a_{q,s}\}) \Rightarrow Match(a_{p,r}, a_{q,s})$

Heuristic 6.4.5 is applicable when we have two attributes in two primary keys which each of them is the single attribute appearing at the left side of a FD. In this case, we match these attributes with each other. We also match the attributes on the right sides of the two FDs if there is only one attribute appearing at the right side of them (heuristic 6.4.6).

Using heuristic 6.4.5 for Example 6.4.1, we match *author* with *name* which is a right decision. We do this because of the two FDs: $author \rightarrow institution$ and $name \rightarrow affiliation$. We also match *institution* with $affiliation$ which are the only attributes appearing at the right side of these FDs based on heuristic 6.4.6.

6.4.7. HEURISTIC. Let PK_p and $PK_q, p \neq q$, be the primary keys of S_p and S_q respectively. Then,

$$(\forall a_{p,r} \in PK_p \backslash \{a_{p,i}\}, \exists a_{q,s} \in PK_q \backslash \{a_{q,j}\} \mid Match(a_{p,r}, a_{q,s})) \wedge \\ (|PK_p| = |PK_q|) \Rightarrow Math(a_{p,i}, a_{q,j})$$

Heuristic 6.4.7 is applicable when all attributes of PK_p and PK_q have been matched, and only one attribute is left in each of them. In such case we match these two attributes with each other hoping that they are semantically the same. Coming back to Example 6.4.1, there is only one attribute left in each of the primary keys that we have not yet matched (i.e. *year*, *date*) that we can match using this heuristic.

Distance Function Algorithm

Algorithm 8 describes how we combine the attributes' name similarity and FD heuristics to build the distance function. Steps 2-12 of the algorithm apply heuristic 6.4.3. They look for FD pairs from different sources which their left sides match together and then try to match attribute pairs on the right sides of these FDs. After finding such FDs, steps 5-10 repeatedly find the attribute pairs (a_p, a_q) whose similarity is maximum. If the similarity of a_p and a_q is more than threshold t_R, their distance is set to *MD* (Match Distance), and the distances between each of them and any other source-mates are set to *UMD* (Unmatch Distance). The algorithm uses the $DoMatch$ procedure for matching and unmatching attributes. It gets the attributes which should be matched as parameter, matches them, and unmatches every one of them with the other ones' source-mates. Generally, whenever the algorithm matches two attributes with each other, it also unmatches the two of them with the other one's source-mates because every attribute of a source can be matched with at most one attribute of every other source. Steps 9 remove the matched attributes from the list of unmatched attributes.

The $IsMatch$ function, which is used by step 3, takes as parameter the left sides of two FDs and returns true if they can be matched together, otherwise it returns false. It first checks whether the input parameters are two sets of the same size. Then, it finds the attribute pair with maximum name similarity and treats it as matched pair by removing the attributes from the list of unmatched attributes if their similarity is more than threshold t_L. It repeats the matching process until there is no more attribute eligible for matching. After the matching loop is over, the function returns true if all attribute pairs have been matched together, otherwise it returns false which means the matching process has not

Algorithm 8 Distance Function

Input:
- Source schemas $S_1, \ldots, S_n$
- $\{F_1, \ldots, F_n\}$ the sets of FDs (the FDs related to PK are omitted)
- $P = \{PK_1, \ldots, PK_n\}$ the set of primary keys of all sources

Output: Distance matrix $D[m][m]$

compute $A = \{a_1, \ldots, a_m\}$ the set of all source attributes
for all FD pair $fd_i \in F_k, fd_j \in F_l, k \neq l$ **do** // heuristic 6.4.3
  if $IsMatch(L_i, L_j)$ **then**
    make local copies of fd_i, fd_j
    repeat
      find the attribute pair $a_p \in R_i, a_q \in R_j$ with the maximum similarity s
      if $s > t_R$ **then**
        $DoMatch(a_p, a_q)$
        $R_i \leftarrow R_i \backslash \{a_p\}; R_j \leftarrow R_j \backslash \{a_q\}$
    until $s > t_R$ and $|R_i| > 0$ and $|R_j| > 0$
for all pair $PK_i, PK_j \in P$, where they are PKs of S_i and S_j respectively **do**
  make local copies of PK_i and PK_j
  for all pair $a_p \in PK_i, a_q \in PK_j$ **do**
    if $\exists fd_k \in F_i$ and $fd_l \in F_j$ such that $L_k = \{a_p\}$ and $L_l = \{a_q\}$ **then**
      $DoMatch(a_p, a_q)$ // heuristic 6.4.5
      $PK_i \leftarrow PK_i \backslash \{a_p\}$; $PK_j \leftarrow PK_j \backslash \{a_q\}$
      if ($R_k = \{a_s\}$ and $R_l = \{a_t\}$) **then** $DoMatch(a_s, a_t)$ // heuristic 6.4.6
  repeat
    find the attribute pair $a_p \in PK_i$ and $a_q \in PK_j$ with maximum similarity s
    if $s > t_{PK}$ **then**
      $DoMatch(a_p, a_q)$ // heuristic 6.4.4
      $PK_i = PK_i \backslash \{a_p\}$; $PK_j = PK_j \backslash \{a_q\}$
  until $s > t_{PK}$ and $|PK_i| > 0$ and $|PK_j| > 0$
  if ($PK_i = \{a_p\}$ and $PK_j = \{a_q\}$) **then** $DoMatch(a_p, a_q)$ // heuristic 6.4.7
for all attribute pair $a_i, a_j \in A$ which $D[a_i][a_j]$ has not been computed yet **do**
  if $(a_i, a_j \in S_k)$ **then** $D[a_i][a_j] \leftarrow$ UMD // heuristic 6.4.2
  else $D[a_i][a_j] \leftarrow similarity(a_i, a_j)$
$\forall a_i, a_j, a_k \in A$ **if** ($D[a_i][a_k] =$ MD and $D[a_k][a_j] =$ UMD) **then** $D[a_i][a_j] \leftarrow$ UMD
$\forall a_i, a_j, a_k \in A$ **if** ($D[a_i][a_k] =$ MD and $D[a_k][a_j] =$ MD) **then** $D[a_i][a_j] \leftarrow$ MD
$\forall a_i, a_j \in AD[a_i][a_j] \leftarrow D[a_j][a_i]$

been successful. Notice that we do not reflect the matching of attributes of the left sides of FDs in the distance matrix. The reason is that for these attributes (in contrast to those on the right side), the matching is done just based on attribute name similarity and not the knowledge in FDs.

Please notice that we use three different similarity thresholds (i.e. t_L, t_R, and t_{PK}) to have more flexibility in the matching. However, we need to set them

carefully. If we set them to high values, we prevent wrong matching but may miss some pairs that should have been matched. On the other hand, if we set thresholds to low values, we increase the number of correctly matched pairs but also increase the number of wrongly matched pairs. In other words, setting the threshold values is a trade off between precision and recall. Aside from this, the inequality between them is important as we explain below. We know that t_L is the similarity threshold for matching attributes at the left sides of FDs. Since the matching of left sides of FDs is taken as evidence for matching the right sides of them, t_L needs to be chosen carefully. Setting it to low values, results in wrong matchings. On the other hand, we use t_R as similarity threshold for matching attributes on the right sides of FDs. Since we already have evidence for matching them, we can be more relaxed in setting t_R by setting it to values lower than t_L. The same argument goes for the value of t_{PK}. t_{PK} is the similarity threshold for matching PK attributes. Since these attributes are a subset of source attributes, it is reasonable to set t_{PK} to lower values than t_L and t_R.

In steps 11-24, we apply heuristics 6.4.4, 6.4.5, 6.4.6 and 6.4.7. Steps 13-17 check every attribute pair of two PKs to see if they are the only attributes at the left sides of two FDs. If yes, then these attributes are matched together. Steps 18-23 find the attribute pair with the maximum name similarity and if it is more than threshold t_{PK}, the attributes are matched together. The matching process continues until there is at least one attribute in every PK and the similarity of the attribute pair with the maximum similarity is more than threshold t_{PK}. If each of the two PKs has only one attribute left, step 24 matches them together based on heuristic 6.4.7.

In steps 25-27, we set the distances of attribute pairs which have not been set yet. Step 26 applies heuristic6.4.2 by setting the distance of the attributes from the same source to *UMD*. The distance of other attribute pairs is set to their name similarity. Steps 28-29 perform a transitive closure over the match and unmatch constraints. Step 30 deals with the symmetric property of the distance function to ensure that the returned distance is independent from the order of attributes.

The matching and unmatching decisions made by a distance function should be consistent with each other. More precisely, a consistent distance function should not satisfy the following condition:

$$\exists a_i, a_j, a_k \in A \mid match(a_i, a_j) \wedge match(a_j, a_k) \wedge unmatch(a_i, a_k). \tag{6.2}$$

The following proposition shows that our distance function is consistent.

6.4.8. PROPOSITION. *Algorithm 8 returns a consistent distance function.*

Proof. We first show that if inconsistency exists, it is removed by step 32 of the algorithm, i.e. the first transitive closure rule. Then, we show that order of applying the transitive closure rules in Algorithm 8 is the only correct order.

Let us prove the first part. Suppose steps 1-31 of the algorithm create an inconsistency so that condition (6.2) satisfies. Then, as the result of step 32 of the algorithm, either $match(a_i, a_j)$ changes to $unmatch(a_i, a_j)$ or $match(a_j, a_k)$ changes to $unmatch(a_j, a_k)$. It is clear that the inconsistency between a_i, a_j, and a_k is removed with either of the changes. Without the loss of generality, we assume that $match(a_i, a_j)$ changes to $unmatch(a_i, a_j)$. Then, if there exists $a_l \in A$, so that condition (6.2) satisfies for a_i, a_j, and a_l as a result of the change, step 32 removes it too. Thus, step 32 removes all of the inconsistencies in the cost of losing possibly correct match pairs.

Let us prove the second part. Suppose that steps 1-31 of the algorithm create an inconsistency so that condition (6.2) satisfies and we change the order of transitive closure rules. By first applying rule 2, $unmatch(a_i, a_k)$ changes to $match(a_i, a_k)$. However, we already unmatched a_i with a_k as the result of matching a_i with one of the source-mates of a_k, say a_l. Thus, we have: $match(a_k, a_i)$ and $match(a_i, a_l)$, which results in $match(a_k, a_l)$ by applying rule 2 to them. This means that we matched two attributes a_k and a_l from the same source. Thus, changing the order of transitive closure rules does not remove the inconsistency but propagates it. □

6.4.2 Schema Matching Algorithm

We now describe the schema matching algorithm which works based on the CAHC clustering method. We use the distance function to compute the distance between clusters in the CAHC clustering method. Algorithm 9 describes how we create probabilistic mediated schemas. This algorithm takes as input the source schemas, distance matrix, and the needed number of mediated schemas (k) which is specified by the user. Steps 1-2 create the first mediated schema by putting every attribute in a cluster. The algorithm stores all created mediated schemas in the set M, and so does for the first created mediated schema in step 3.

Steps 4-11 repeatedly find the two clusters with the minimum distance while the distance between two clusters is defined as follows: if the clusters have two attributes from the same source, the distance between them is infinity; otherwise the minimum distance between two attributes, each from one of the two clusters, is regarded as the distance between the two clusters. These clusters are merged together by step 9 if their distance is not equal to infinity, and the newly created mediated schema is added to M by step 10. We consider the infinity as the minimum distance between clusters when every two clusters have attributes from the same source. In such a case, we stop creating the mediated schemas.

To assign a rank to each generated mediated schema, we define FD-point as the number of matched pairs which has been respected by a mediated schema. For every created mediated schema, Step 12 computes its FD-point, which is a metric for measuring the quality of mediated schemas and for selecting only the high quality ones. Distance matrix recommends some attribute pairs to be

Algorithm 9 Schema Matching

Input:
- Source schemas $S_1, \ldots, S_n$
- Distance matrix $D[m][m]$
- Number of needed mediated schemas k

Output: A set of probabilistic mediated schemas

1: compute $A = \{a_1, \ldots, a_m\}$ the set of all source attributes
2: let C be the set of clusters c_i such that $c_i = \{a_i\}, a_i \in A, i \in [1, m]$
3: $M \leftarrow C$
4: **repeat**
5: find two clusters $c_i, c_j \in C$ having the minimum distance d_{min} while distance d_{ij} between c_i and c_j is computed as follows:
6: **if** $(\exists a_k \in c_i, a_l \in c_j, a_k, a_l \in S_p)$ **then** $d_{ij} \leftarrow \infty$
7: **else** $d_{ij} \leftarrow Min(D[a_k][a_l]), a_k \in c_i, a_l \in c_j$
8: **if** $d_{min} \neq \infty$ **then**
9: merge c_i with c_j
10: add the newly added mediated schema to M
11: **until** $d_{min} \neq \infty$
12: **for each** $C_i \in M$ compute the $FDpoint_i$ as the number of attribute pairs recommended by distance matrix and respected by C_i
13: $FDpoint_{max} \leftarrow Max(FDpoint_i), C_i \in M$
14: $M \leftarrow \{C_i \mid C_i \in M, FDpoint_i = FDpoint_{max}\}$
15: **if** $k < |M|$ **then**
16: select k mediated schemas randomly from M
17: assign probability $\frac{1}{k}$ to every selected mediated schema and return them
18: **else** assign probability $\frac{1}{|M|}$ to every $C_i \in M$ and return them

put in the same cluster by returning their distance as *MD*. FD-point is defined as the number of these recommendations which are respected by the mediated schema. Steps 13-14 select the mediated schemas with the maximum FD-point. We call them as eligible mediated schemas. Steps 15-18 return k randomly selected eligible mediated schemas to the user. Since the algorithm has no means for differentiating between eligible mediated schemas, it assigns equal probabilities to all returned mediated schemas.

6.4.3 Adding Data Sources Incrementally

IFD starts with a given set of sources and ends up generating several mediated schemas from these sources. A useful property of IFD is that it allows new sources to be added to the system on the fly. Let S_{n+1} be the source which we want to add. By comparing S_{n+1} with each $S_i, i \in [1..n]$, we can compute the distance between every attribute of S_{n+1} and every attribute of S_i in the same way that we did

for computing the distances between the attributes of $S_1..S_n$. After computing the distances, we consider every PMS, say $M_j, j \in [1..k]$ and for every attribute $a_p \in S_{n+1}$, we find the closest attribute $a_q \in A$ and put a_p in the same cluster as that of a_q. We repeat this process for every PMS.

This is a useful property of IFD which is needed in the contexts which we do not have all sources at hand when we start setting up the data integration application and we need to add them incrementally when they become available.

6.4.4 Execution Cost Analysis

In this section, we study the execution costs of our schema matching and distance function algorithms.

6.4.9. THEOREM. *Let m be the number of the attributes of all sources, then the running time of Algorithm 8 and the schema matching algorithm together is $\theta(m^3)$.*

Proof. The basis for our schema matching algorithm is the single-link CAHC (Constrained Agglomerative Hierarchical Clustering) algorithm in which the number of clusters is determined by the arity of the source with the maximum arity. Let m be the number of the attributes of all sources. The time complexity of the single-link AHC algorithm implemented using next-best-merge array (NBM) is $\Theta(m^2)$ [90].

Let us now analyze the running time of the distance function algorithm. Most of the algorithm is devoted to matching left and right sides of FDs, or the attributes of PKs. Let c be the arity of the source with the maximum arity, and f the maximum number of FDs that a source may have, which is a constant. The number of attributes on the left and right side of a FD is at most equal to the arity of its source, so its upper bound is c. Thus, matching both sides of two FDs takes $\Theta(c^2)$ time which is equal to $\Theta(1)$ because c is a constant. This argument also holds for matching PKs' attributes because the algorithm only checks the FDs of the two sources (which each one at most has f FDs), not the FDs of all sources.

Let n be the number of sources, then we have at most $f \times n$ FDs. The algorithm checks every FD pair for matching. Thus, it takes $\frac{f \times n \times (f \times n - 1)}{2} \times \Theta(1)$ time for matching FDs which is equal to $(n^2 \times f^2)$. By taking f, i.e. the maximum number of FDs, as a constant, the complexity is $\Theta(n^2)$. In the same way, the time complexity for matching PKs is $\Theta(n^2)$.

The transitive closure part of the algorithm is done in $\theta(m^3)$ time, where m is the total number of attributes. The last part of the algorithm that guarantees symmetric property takes $\theta(m^3)$. Since the number of attributes is at least the number of sources, we have $m \geqslant n$. Thus, the transitive closure of attributes dominates all other parts of the algorithm and the running time of the algorithm

is $\theta(m^3)$. As a result, the running time of the schema matching and the distance function algorithms together is $\theta(m^3)$. □

6.5 Performance Evaluation

In this section, we study the effectiveness of our data integration solution. In particular, we show the effect of using functional dependencies on the quality of generated mediated schemas. We compare our solution with the one presented in [115] which is the closest to ours. To examine the contribution of using a probabilistic approach, we compare our approach with two traditional baseline solutions that do not use probabilistic techniques, i.e. they generate only one single deterministic mediated schema.

The rest of this section is organized as follows. We first describe our experimental setup. Then, we compare the performance of our solution with the competing approaches.

6.5.1 Experimental Setup

We implemented our system (IFD) in Java. We took advantage of Weka 3-7-3 classes [74] for implementing the hierarchical clustering component. We used the SecondString Java package[2] to compute the Jaro Winkler similarity [139] of attribute names in pair-wise attribute comparison. We conducted our experiments on a Windows XP machine with Intel core 2 GHz CPU and 2GB memory.

In our experiments, we set the number of mediated schemas (denoted as n) to 1000, which is relatively high, in order to return all eligible mediated schemas. Our experiments showed similar results when we varied n considerably (e.g. n = 5). The default values for the parameters of our solution are as follows. We set similarity threshold for PK attributes (t_{PK}) to 0.7, similarity threshold for attributes on the left side of functional dependencies (t_L) to 0.9, similarity threshold for attributes on the right side of functional dependencies (t_R) to 0.8, the distance between attributes being matched (MD) to 0, and the distance between attributes being unmatched (UMD) to 1.

We evaluated our system using a dataset in the university domain. This dataset consists of 17 single-table schemas which we designed ourselves. For having variety in attribute names, we used Google Search with "computer science" and "course schedule" keywords and picked up the first 17 related results. For every selected webpage, we designed a single-table schema which could be the data source of the course schedule information on that webpage and we used data labels as attribute names of the schema. Also, we created primary key and functional dependencies for every schema using our knowledge of the domain. This dataset, denoted by *Courses*, can be found in Appendix B.

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

To evaluate the quality of generated mediated schemas, we tested them against the mediated schema which we created manually. Since each mediated schema corresponds to a clustering of source attributes, we measured its quality by computing the precision, recall, and F-measure of the clustering. We computed the metrics for each individual mediated schema, and summed the results weighted by their respective probabilities.

To the best of our knowledge, the most competing approach to ours (IFD) is that of Sarma et al. [115] which we denote by UDI as they did. Thus, we compare our solution with UDI as the most competing probabilistic approach. We implemented UDI in Java. We used the same tool in our approach for computing pair-wise attribute similarity as in UDI. Also, we set the parameters edge-weight threshold and error bar to 0.85 and 0.02 respectively. Since the time complexity of UDI approach is exponential to the number of uncertain edges, we selected the above values carefully to let it run.

To examine the performance gain of using a probabilistic technique, we considered two baseline approaches that create a single mediated schema:

- FD1: creates a deterministic mediated schema as follows. In Algorithm 9, we count the number of FD recommendations and obtain the maximum possible FD-point, then we stop at the first schema which gets this maximum point.

- SingleMed: creates a deterministic mediated schema based on Algorithm 4.1 in [115]. We set frequency threshold to 0 and the edge weight threshold to 0.85.

Also, to evaluate the contribution of using functional dependencies in the quality of generated mediated schemas, we considered Algorithm 9 without taking advantage of the FD recommendations (WFD) and compared it to our approach.

6.5.2 Results

Quality of Mediated Schemas

In this section, we compare the quality of mediated schemas generated by our approach (IFD) with the ones generated by UDI and other competing approaches.

Figure 6.2 compares the results measuring precision, recall, and F-measure of IFD, UDI, Single-Med, FD1, and WFD. It shows that IFD obtains better results than UDI. It improves precision by 23%, recall by 22%, and F-measure by 23%.

Figure 6.2 also shows the contribution of using FD recommendations in the quality of the results. WFD (Without FD) shows the results of our approach without using FD recommendations. It is obvious that using these recommendations has considerable effect on the results.

Furthermore, Figure 6.2 shows the performance gain of using a probabilistic approach rather than a single deterministic schema approach. FD1 applies all of

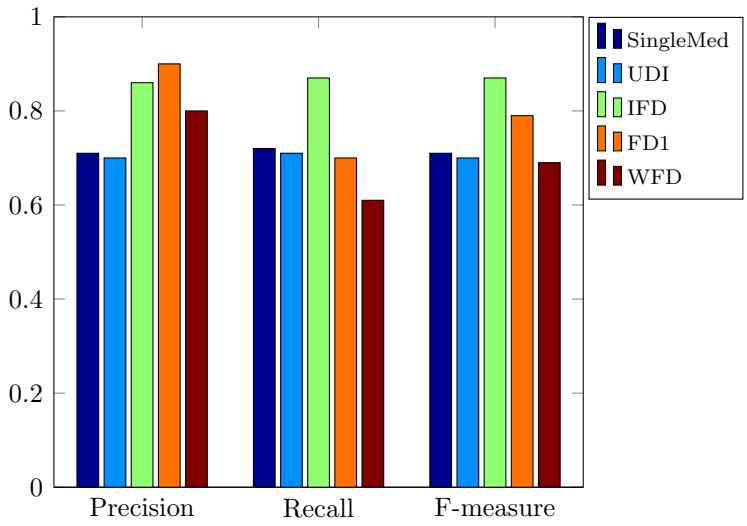


Figure 6.2: Performance comparison of IFD with competing approaches

the FD recommendations to obtain the mediated schema with the maximum FD-point, then stops and returns the resulted mediated schema. On the other hand, IFD does not stop after applying all FD recommendations but since there is no further FD recommendation, it starts merging clusters based on the similarity of their attribute pairs. This increases recall considerably, but reduces precision a little because some pairs are clustered wrongly. Overall, IFD improves F-measure by 8% compared to FD1. On the other hand, this Figure shows that UDI does not get such performance gain compared to Single-Med which creates a single deterministic schema. This happens because UDI cannot select the high quality schemas among the generated schemas.

Scalability

To investigate the scalability of our approach, we measure the effect of the number of sources (n) on its execution time. By execution time, we mean the setup time needed to integrate n data sources. For IFD, the execution time equals to the execution time of computing distances using Algorithm 8 plus the execution time of generating mediated schemas using Algorithm 9. For UDI, we only consider the time needed to generate mediated schemas to be fair in our comparison. For UDI, the execution time is the time needed to create the mediated schemas.

Figure 6.3 shows how the execution times of IFD and UDI increase with increasing n up to 17 (the total number of sources in the tested dataset). The impact of the number of sources on the execution time of IFD is not as high as that of UDI. While in the beginning, the execution time of UDI is a little lower than IFD, it dramatically increases eventually. This is because the execution time of IFD is cubic to the number of the attributes of sources (see Section 6.4.4), but

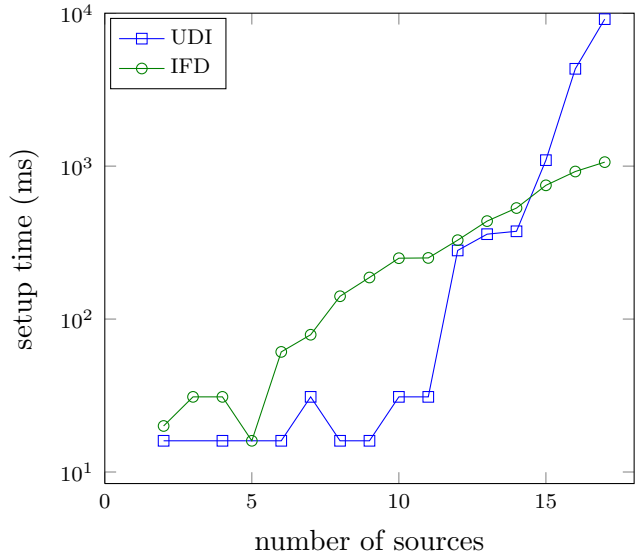


Figure 6.3: Execution time comparison of IFD and UDI

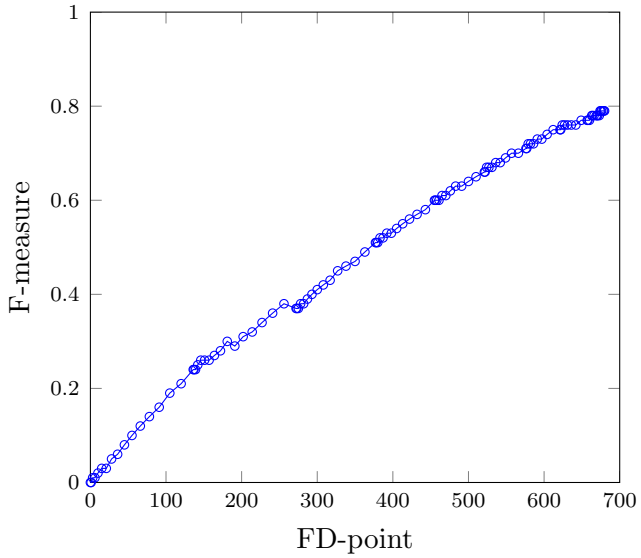


Figure 6.4: Effect of FD-point on F-measure in IFD approach

that of UDI is exponential to the number of uncertain edges. This shows that IFD is much more scalable than UDI.

Effect of FD-point

In this section, we study the effect of FD-point on F-measure. Figure 6.4 shows how F-measure increases with increasing FD-point up to 680 which is the maximum possible value in the tested dataset. The starting point is when we have

one cluster for every attribute. We have not used any recommendation at this point yet; as a result, *FD-point* = 0. Also it is clear that *precision* = 1 and *recall* = 0, thus *F-measure* = 0. As we begin merging clusters using recommendations, FD-point increases and this increases the F-measure as well. The increase in FD-point continues until it reaches its maximum possible value in the tested dataset. We consider all generated mediated schemas with maximum FD-point value as schemas eligible for being in the result set.

6.6 Analysis against related Work

There has been much work in the area of automatic schema matching during the last three decades (see [108] for a survey). They studied how to use various clues to identify the semantics of attributes and match them. An important class of approaches, which are referred to by constraint matchers, uses the constraints in schemas to determine the similarity of schema elements. Examples of such constraints are data types, value ranges, uniqueness, optionality, relationship types, and cardinalities. For instance, OntoMatch [26] and DIKE [101] use this type of matcher. Our approach is different, since we use an uncertain approach for modeling and generating mediated schemas. Thus, the heuristic rules that we use as well as the way we decrease the distance of the attributes, is completely different. In addition, we take advantage of FDs. The proposals in [32] and [84] also consider the role of FDs in schema matching. However, our heuristic rules and the way we combine it with attribute similarity is completely different than these proposals.

The closest work to ours is that of Sarma et al. [115] which we denoted as UDI in this chapter. UDI creates several mediated schemas with probabilities attached to them. To do so, it constructs a weighted graph of source attributes and distinguishes two types of edges: certain and uncertain. Then, a mediated schema is created for every subset of uncertain edges. Our approach has several advantages over UDI. The time complexity of UDI's algorithm for generating mediated schemas is exponential to the number of uncertain edges (i.e. attribute pairs) but that of our algorithm is PTIME (as shown in Section 6.4.4), therefore our approach is much better scalable. In addition, the quality of mediated schemas generated by our approach has shown to be considerably higher than that of UDI. Furthermore, the mediated schemas generated by our approach are consistent with all sources, while those of UDI may be inconsistent with some sources.

6.7 Conclusion

In this chapter, we proposed IFD, a data integration system with the objective of automatically setting up a data integration application. We established an

advanced starting point for pay-as-you-go data integration systems. IFD takes advantage of the background knowledge implied in FDs for finding attribute correlations and using it for matching the source schemas and generating the mediated schema. We built IFD on a probabilistic data model in order to model the uncertainty in data integration systems.

We validated the performance of IFD through implementation. We showed that using FDs can significantly improve the quality of schema matching (by 26%). We also showed the considerable contribution of using a probabilistic approach (causing an improve by 10%). Furthermore, we showed that IFD outperforms UDI, its main competitor, by 23% and has cubic scale up compared to UDI's exponential execution cost.

Chapter 7

Conclusion

In this chapter, we first revisit the research questions posed in the introduction and discuss how we answered them in our research work presented in Chapters 3 to 6. We then discuss a number of possible extensions and future work for the presented research of this dissertation.

7.1 Answers to Research Questions

The first question posed in the introduction of this thesis was the following.

1. *How to match a probabilistic entity against a set of probabilistic entities, while considering both their similarity and probability? In other words, what are the semantics of identity resolution problem over probabilistic data?*

We addressed this question in Chapter 3. We adapted the possible worlds semantics of uncertain data to define the novel concepts of *most-probable matching pair* (MPMP) and *most-probable matching entity* (MPME), together referred to as *most-probable matches*.

Chapter 3 also dealt with the second posed question, which was:

2. *How can we efficiently deal with the identity resolution problem over probabilistic data?*

To propose an efficient solution for computing the most-probable matches, in Chapter 3 we differentiated between two classes of similarity functions: i.e. *context-free* and *context-sensitive*. For context-free similarity functions, we proposed the CFA algorithm, which simultaneously computes MPMP and MPME concepts in PTIME. Moreover, we proposed two optimized versions of the CFA algorithm, i.e. CFA-MPMP and CFA-MPME, which respectively compute MPMP

and MPME concepts. For context-sensitive similarity functions, we used the Monte-Carlo approximation algorithm. To speedup the Monte-Carlo algorithm, we proposed a parallel version of it using the MapReduce framework. Moreover, to overcome the high response time of most context-sensitive similarity functions in the literature, which makes them very inefficient for the Monte-Carlo algorithm, we proposed the novel CB similarity function with the following salient features:

- CB is very efficient compared to other context-sensitive similarity functions because it significantly reduces the number of rather costly string comparison operations by working at the attribute level, rather than at the word or q-gram level.
- In contrast to most of the tuple matching methods that work at the attribute level, CB is *self-tuning*, meaning that it does not need the specification of weights for representing the relative importance of individual attributes.

The third posed research question was the following.

3. *How can we efficiently deal with the identity resolution problem over probabilistic data in distributed systems?*

Chapter 4 deals with this question. In this chapter, we proposed the FD, a fully distributed algorithm for dealing with the identity resolution problem over distributed probabilistic data, with the objective of minimizing network traffic. FD uses the novel concepts of *potential* and *essential-set* to prune data at local nodes. This leads to a significant reduction in network traffic and response time compared to the baseline approaches. FD requires no global information, and does not depend on the existence of certain nodes.

Chapter 5 dealt with the fourth posed question, which was:

4. *Does deduplication necessarily improve the quality of a probabilistic database? If not, then how can we improve the quality of a probabilistic database through deduplication?*

In Chapter 5, we used entropy as a quality metric for measuring the quality of a probabilistic database, where the higher the entropy of a database, the lower is its quality. We showed that if entropy is not taken into account, deduplication does not necessarily improve the quality of a probabilistic database. Thus, to guarantee the quality improvement, we modeled deduplication problem over probabilistic data as an entropy minimization problem. We then proposed an efficient solution for the deduplication problem in probabilistic data as follows:

- We proposed an efficient technique for computing the entropy of a probabilistic database in the x-relation probabilistic data model [10].

- We proposed a merge function for merging x-tuples in the x-relation data model.

- The properties of our proposed merge function, i.e. commutative and associative, as well as entropy-reduction properties, enabled us to propose a PTIME algorithm for dealing with the deduplication problem, and producing a cleaned database with (near) minimum entropy.

The last research question was:

5. *How effectively can we deal with the schema matching problem in a fully automated setting?*

We answered this question in Chapter 6, where we dealt with the schema matching problem in setting up a fully automated data integration system, denoted by the IFD, from a number of heterogenous data sources. IFD has two important features which allow it to effectively deal with the schema matching problem. First, IFD is built on a probabilistic data model in order to capture the uncertainty that arises during the schema matching process. Second, IFD takes advantage of the background knowledge which is implied in functional dependencies for finding attribute correlations and using it for matching the source schemas. We showed that it is possible to achieve a fairly high accuracy in dealing with the schema matching problem in a fully automated setting. This lets us to effectively deal with the entity resolution problem when data resides in heterogenous data sources.

Having answered the sub-questions, we come back to the main research question of this thesis.

How can we, effectively and efficiently, deal with the entity resolution problem for probabilistic data?

As discussed in Chapter 2, dealing with the entity resolution problem, both on deterministic and probabilistic data, greatly depends on the used similarity function. Our proposed methods for the entity resolution problem over probabilistic data is generic, which can thus be applied to any similarity function, suitable for the application in hand. On the other hand, to efficiently deal with the entity resolution problem over probabilistic data, we need to avoid enumerating the possible worlds of uncertain data. Our efficient methods heavily rely on the properties of the x-relation data model, which results in the PTIME time complexity of all of our proposed techniques, except the Monte-Carlo algorithm in Section 3.4. Exploiting the properties of other probabilistic data models for efficient handling of entity resolution problem over them however, remains as a possible direction for future research.

7.2 Future Work

While this thesis has made a number of contributions to the problem of entity resolution over probabilistic data, the general problem still is open. This work, in fact, opens the following directions for future research.

ERPD over other data models. First, as discussed in previous section, while the defined semantics of the ERPD problem is not specific to any probabilistic data model, we relied on the properties of the x-relation probabilistic data model for efficient dealing with this problem. Efficient entity resolution over other probabilistic data models however, or even other uncertain data models, is one possible direction for future research.

Using CB in blocking methods. Due to its efficiency and effectiveness, we believe that our CB similarity function presented in Chapter 3, can act as a cheap similarity metric in the blocking methods, aiming to further improve the efficiency of ER (see Section 2.2.2). Elaborating on this idea is another possible direction for future research.

ERPD for special distributed systems. In Chapter 4, we assume a very general topology for distributed system. However, in some applications, probabilistic data might be fragmented over a distributed system with a particular topology, e.g. *distributed hash tables*. Attacking the ERPD problem in such distributed systems is a possible future research direction.

Using functional dependencies in other schema matchers. Finally, in Chapter 6, we showed that using the background knowledge implied within functional dependencies alone can significantly improve the quality of schema matching. On the other hand, there exist many schema matchers in the literature that use other features, e.g. data types and value ranges, to match the schemas. Integrating other existing high quality schema matchers with our schema matching heuristics is another possible direction for future research.

Appendix A

Uncertain Data Models

In this chapter, we review a number of uncertain data models. In the first three sections, we restrict our attention to the uncertain relational models, then in Section A.4, we explain some other models.

A.1 Incomplete Relational Models

The early uncertain data models mostly focus on representing incomplete information without caring about their possible existence probabilities. As in the literature, we refer to these models as incomplete data models. The notion of *null* values is perhaps the earliest attempt to model the incomplete information in the relational model. In this section, we begin by applying this way of modeling uncertain data, and then we present the more expressive incomplete relational models.

A.1.1 Codd tables

The aim of *Codd tables*, proposed by Codd [47, 48], is to model missing information in relational tables. The idea is to represent missing attribute values by a symbol, called *null*, which is different from all other data values. Let for instance symbol $\perp$ denote the null. For illustration, consider the Codd table T, shown in Figure A.1(a), where the missing age of Mary and John, and Bob's phone number are represented using null values.

Codd has adopted the three-valued logic for evaluating the queries over Codd tables. According to this logic, there are three logical values indicating *true*, *false*, and *unknown*. Besides the rules of the boolean logic, three-valued logic has a number of additional rules, which are shown in Figure A.1(b). In the Codd's proposed query evaluation semantics, comparing any value (including null itself) with null results in the *unknown* logical value. This semantics is currently in use in most commercial database management systems. However, this approach

	name	**age**	**phone-no**
t_1	Bob	35	$\perp$
t_2	Mary	$\perp$	789526
t_3	John	$\perp$	5256661

(a)

$\wedge$	unknown
true	unknown
false	false
unknown	unknown

$\vee$	unknown
true	true
false	unknown
unknown	unknown

$\neg$ unknown = unknown

(b)

Figure A.1: a) Table T: an example Codd table, b) The additional rules in three-valued logic

may return counter-intuitive results [66]. For example, consider evaluating the following query over table T, shown in Figure A.1(a):

SELECT * FROM T
WHERE $age = 25$ OR $age \neq 25$

The query returns t_1 as the result, while one might expect all tuples in T as the query result [66].

Codd tables are very limited in representing incomplete information. They can only represent the fact that some attribute values are missing, and cannot represent any other kind of additional information about the missing attribute values. Consider, for instance, the case that we do not know the ages of Mary and Bob, but we know that Mary is younger than Bob. We cannot represent this kind of information using Codd tables. On the other hand, Codd tables are closed only under *selection* and *projection* operations [78]. To overcome the limitations of Codd tables, *c-tables* (short for *conditional tables*) [78] have been proposed.

A.1.2 C-tables

A c-table can represent the uncertainty at both the attribute and tuple level. At the attribute level, it allows constants and variables as attribute values, and at the tuple level, a logical expression, indicating the existence condition of the tuple in the table, is associated with each tuple. A possible world is built by an instantiation of variables with constants, where each possible world contains the tuples whose condition are satisfied by the instantiation. For illustration, Figure A.2 shows a c-table.

	name	**age**	**phone-no**	**condition**
t_1	Bob	35	x	true
t_2	Mary	y	789526	$z > y$
t_3	John	z	5256661	$z > y \wedge z \in [30..40]$

Figure A.2: An example c-table

	A	**B**	**C**
t_1	x	1	0
t_2	0	y	x
t_3	y	2	3

(a) e-table

	A	**B**	**C**
t_1	x	1	0
t_2	0	y	z
t_3	v	2	3

$x \neq z \wedge y \neq 0$

(b) i-table

	A	**B**	**C**
t_1	x	1	0
t_2	0	y	x
t_3	y	2	3

$x \neq 0$

(c) g-table

Figure A.3: Examples of e-table, i-table, and g-table

The c-tables model is a complete model, and closed under the whole relational algebra database language. The query evaluation however is computationally expensive [9]. Moreover, the use of variables in the model makes it hard for users to read and reason with it [107, 117]. While the c-tables model is a very powerful formalism, these shortcomings make it an impractical uncertainty model.

A number of variations of c-tables have been proposed in the literature. The c-table associated with a global condition has been proposed in [65]. Furthermore, removing the tuples' conditions from c-table, Abiteboul et al. [9] has proposed the following three models: *e-table* that is a table where only the equality of variables are allowed; *i-table* that is a table with a global condition, where the condition is restricted to a conjunction of inequalities on variables; and *g-table* that is a combination of e-table and i-table. For better intuition, Figure A.3 shows examples of these models. To illustrate, repetitive use of variable x in Figure A.3(a) poses the condition $t_1.A = t_2.C$ on the shown e-table. In the i-table, shown in Figure A.3(b), no variable is used repetitively, which thus shows no condition is posed by the variables on the table, but the table is associated with the global condition $x \neq z \wedge y \neq 0$. The combination of repetitive use of variables x and y (representing their equality), and the global condition $x \neq 0$ is shown in the g-table in Figure A.3(c).

A.2 Fuzzy Relational Models

Fuzzy relational models use the *fuzzy set theory* [144] to represent the uncertainty in data. In this section, we first briefly present the concept of fuzzy set, and then

explain how it is used to represent uncertainty in the relational model.

A.2.1 Fuzzy Set

In ordinary set theory, the membership of elements in a set are binary values, meaning that an element either belongs or does not belong to the set. In contrast, the membership of elements in a fuzzy set is represented by real numbers from the range [0..1]. More precisely, a fuzzy set F in a universe U is identified by a membership function $\Phi_F : U \rightarrow [0..1]$, where U is a set of elements, called universe, and $\Phi_F(u)$ for each $u \in U$ denotes the degree of membership of u in the fuzzy set F.

Fuzzy sets mostly are used to represent the imprecise concepts. For instance, the *high salary* concept is an imprecise concept, which might be represented with the fuzzy set *HS*, in universe *Salaries*, with the following membership function:

$$\Phi_{HS}(s) = \begin{cases} 0 & \text{if } s \leqslant 30,000 \\ \frac{1}{1+\lvert s-70,000\rvert} & \text{if } 30,000 < s < 70,000 \\ 1 & \text{if } s \geqslant 70,000 \end{cases}$$

where the universe *Salaries* is the set of integer numbers in range [10,000..100,000].

The fundamental ordinary set operations, i.e. union, intersection, and complement, have fuzzy counterparts defined as follows. Let A and B two fuzzy sets in the universe U with membership function Φ_A and Φ_B, respectively. Then, the membership functions of $A \cup B$, $A \cap B$, and A' are defined as

$$\begin{aligned} \Phi_{A \cup B}(u) &= max(\Phi_A(u), \Phi_B(u)) \\ \Phi_{A \cap B}(u) &= min(\Phi_A(u), \Phi_B(u)) \\ \Phi_{A'}(u) &= 1 - \Phi_A(u) \end{aligned}$$

Let $A_1, \ldots, A_n$ be fuzzy sets in $U_1, \ldots, U_n$, respectively, and $U = U_1 \times \ldots \times U_n$. Then, the cartesian product of $A_1, \ldots, A_n$ is defined to be a fuzzy set in universe U with the following membership function:

$$\Phi_{A_1 \times \ldots A_n}(u_1, \ldots, u_n) = min(\Phi_{A_1}(u_1), \ldots, \Phi_{A_n}(u_n)) \tag{A.1}$$

where $u_i \in U_i$ and Φ_i is the membership function of fuzzy set A_i.

A.2.2 Fuzzy Relations

A fuzzy relation R on schema $S(A_1, \ldots, A_n)$ is a fuzzy set in universe $dom(A_1) \times \ldots \times dom(A_n)$, where $dom(A_i)$ is the domain of attribute A_i, which might itself be an ordinary or a fuzzy set [109]. Similar to ordinary relations, a fuzzy relation can be represented as a table with an additional attribute Φ which shows the

degree of membership of each tuple to the relation. The Φ attribute, for each tuple, is determined using the definition of cartesian product for fuzzy sets, i.e. equation (A.1). The table only contains tuples for which $\Phi > 0$.

The main purpose of a fuzzy relation is to establish a fuzzy association between its attributes using a fuzzy proposition, i.e. a proposition whose truth value can be a number in range [0..1], in contrast to an ordinary proposition whose truth value is either zero or one. To illustrate, consider the fuzzy relation *Likes* in Figure A.4, which establishes a fuzzy association between *person* and *movie* attributes using the fuzzy proposition *"person x likes movie y"*. According to relation *Likes*, the truth value for the fuzzy proposition *"Bob likes the Godfather"* is 0.8.

person	**movie**	**Φ**
John	Heat	0.7
Bob	The Godfather	0.8
Mary	Inception	0.5
Alice	The Truman Show	0.9

Figure A.4: The example fuzzy relation *Likes*

The domain of attributes in the *Likes* fuzzy relation are ordinary sets. However, it is possible to have fuzzy relations of attributes with fuzzy domains. For instance, consider the fuzzy relation R in Figure A.5 which lists movies that are popular and have good screenplays. In this relation, the domain of attribute *movie* is an ordinary set, but those of *popular* and *good-screenplay* are fuzzy sets with the following membership functions:

$$\Phi_{popular}(x) = \frac{x}{10}\,, \qquad x \in \{1, \ldots, 10\}$$
$$\Phi_{good\text{-}screenplay}(y) = \left(\frac{y}{5}\right)^2, \; y \in \{1, \ldots, 5\}$$

The fuzzy relation R can be interpreted as the truth value for the fuzzy proposition *"movie m is popular and has a good screenplay"*. For example, according to R, the truth value for the fuzzy proposition *"The Godfather is popular and has a good screenplay"* is 0.64.

movie	**popular**	**good-screenplay**	**Φ**
The Godfather	9	4	0.64
Heat	8	3	0.36
In Time	6	4	0.6
Rain Man	8	5	0.8

Figure A.5: The example fuzzy relation R

A large number of other fuzzy data models has been proposed in the literature. A discussion of these models is however out of the scope of this thesis, and the interested reader is referred to [36, 37, 35, 128, 129, 105, 106].

A.3 Probabilistic Relational Models

Modeling uncertainty using incomplete data models is not enough for many applications, where, for instance, we need to represent the likelihood of possible worlds or the confidence that we have in a piece of information. This need has been addressed by introducing the probabilistic data models. In these models, an uncertain database, referred to as *probabilistic database*, represents a probability distribution over a set of possible worlds.

ProbView [83] is one of the early probabilistic database systems. In ProbView, each uncertain attribute is represented using a random variable over a set of finite values, and independence is assumed between different random variables. For instance, probabilistic database $\mathcal{D}$, shown in Figure 2.1(a), uses ProbView's data model.

Another early probabilistic data model is that of Fuhr et al. [63]. Fuhr's model manages uncertainty at the tuple level. In this model, each tuple is associated with a probability value indicating the likelihood of its occurrence in the database, and the occurrence of tuples are assumed to be independent from each other. The Fuhr's model has been recently extended to the x-relation data model (see Section 2.1.2).

There are probabilistic models which are a direct extension of an incomplete data model. Consider, for instance, *probabilistic c-tables* [72], where each variable is associated with a discrete probability distribution. As another example, *world set decompositions* [13] have a probabilistic version in which probabilities can be attached to components.

In recent years, a number of probabilistic relational database systems have been developed. The aim of these database systems is to provide built-in support for probabilistic data as first-class citizens, and efficient implementation of uncertain data management concepts and algorithms. These database systems include *MayBMS* [3], *ORION* [4], and *Trio* [8]. MayBMS has been built on top of the PostgreSQL and implements the world set decompositions model. ORION supports both attribute and tuple level uncertainty with arbitrary correlations, and can represent attributes by both discrete and continuous probability distributions. ORION's underlying data model is closed under basic relational algebra operations. Trio implements the x-relation model, and supports data provenance in order to trace the origin of query results.

A.4 Probabilistic Graphical Models

Some uncertain data models combine the relational model with a Probabilistic Graphical Model (PGM for short). In this way, they can capture complex correlations both at tuple and attribute levels, and use well-studied probabilistic reasoning techniques for query evaluation over the uncertain database. *BayesStore* [1] and *PrDB* [7] are two examples of these models. In this section, we present the basics of PGMs.

The aim of a PGM is to efficiently represent and process a joint probability distribution over a set of random variables. To achieve these goals, PGMs extensively rely on exploiting conditional independencies among random variables to compactly encode their joint distribution. In general, a PGM is composed of two components: a graph, and a set functions, also called factors, each over a subset of random variables. The nodes of the graph are random variables, while there exist an edge between the variables that directly interact with each other, and variables with no edge between them are conditionally independent. Depending on the type of the edges between random variables, we can distinguish between two classes of PGMs: directed models, also known as *Bayesian networks*, and undirected models, also known as *Markov networks*.

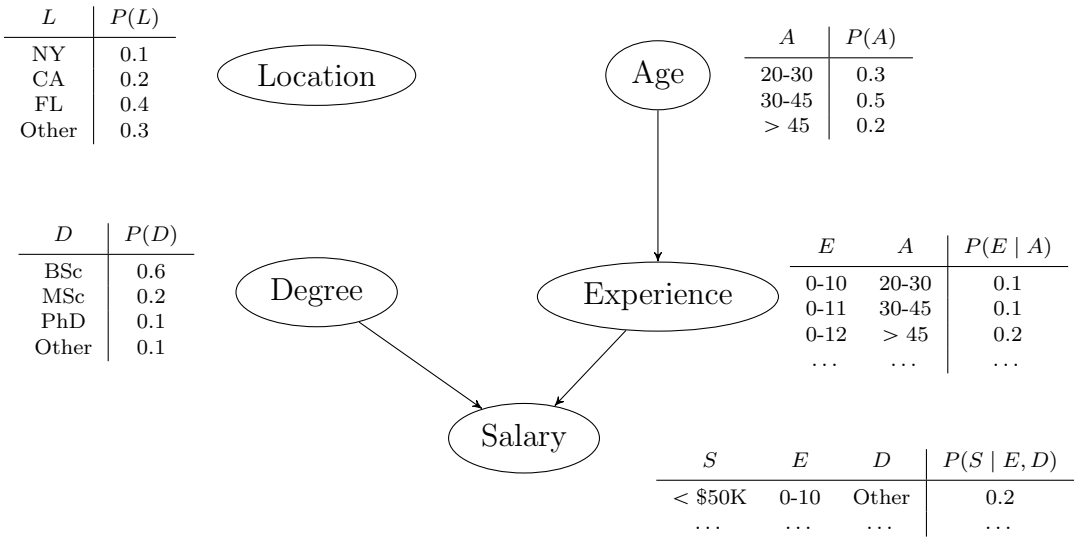
A directed PGM uses a directed acyclic graph to encode three types of dependency between random variables, i.e. dependent, independent, and conditionally independent. In a directed PGM, each variable is dependent to its parents; connected variables (except through an edge) are conditionally independent; and disconnected variables are independent. Each variable X_i is associated with a probability distribution function, denoted by $P(X_i \mid parents(X_i))$, which shows the distribution of probabilities for the values of X_i given the values of its parents. In general, the joint probability distribution that is encoded by a directed PGM is as follows:

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid parents(X_i)),$$

where $X_1, \ldots, X_n$ are random variables.

Let us illustrate directed PGMs using an example from [55]. Figure A.6 shows a bayesian network for modeling the *location*, *degree*, *age*, *experience*, and *salary* of a person. Typically, *location* is independent of all other variables. That is why *location* is not connected to any other attribute. *Age* affects *experience*, and *salary* is affected by both *experience* and *degree*. Although *age* affects *salary*, its effect is indirect through *experience*, meaning that for salary we do not need the *age* of a person when his *experience* is known. Figure A.6 also shows the probability functions, each associated to a variable, and the joint probability distribution that is modeled by the network.

In the undirected PGM, an undirected graph is used to encode the dependency between variables. Let G denote the undirected graph over random variables



$$P(L, D, A, E, S) = P(L)P(A)P(D)P(E \mid A)P(S \mid D, E)$$

Figure A.6: An example Bayesian network (directed PGM)

$X_1, \ldots, X_n$, and $\mathcal{C}$ the set of complete subgraphs of G. Then, the joint probability distribution that is encoded by the undirected PGM is as follows:

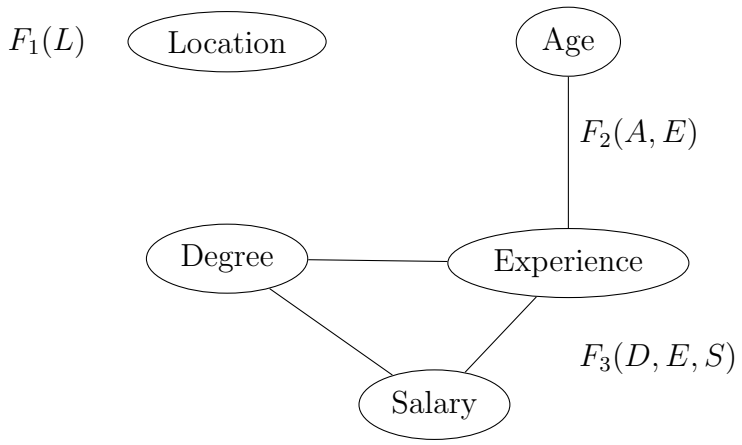
$$P(X_1, \ldots, X_n) = \frac{1}{Z} \prod_{g \in C} F_g(X_g),$$

where for each subgraph $g \in C$, F_g is a probability function over the variables in g, denoted by X_g, and $Z = \sum_X \prod_{g \in C} F_g(X_g)$ is the normalization constant.

To illustrate, consider the example of undirected PGM in Figure A.7, which uses the same variables as our directed PGM example. The complete subgraphs are $\{Location\}$, $\{Age, Experience\}$, and $\{Degree, Experience, Salary\}$, and probability functions are defined on the variables in these subgraphs.

The encoded dependencies by an undirected PGM are as follows: Considering X, Y, and Z as subsets of variables, then, X and Z are conditionally independent given Y if X is separated from Z by Y, where separation, here, means that each path from a variable in X to a variable in Z includes a variable from Y. For instance, Figure A.7 shows that $\{Degree, Salary\}$ and Age are conditionally independent given $Experience$.

In general, directed PGMs are more intuitive and easier to use than undirected PGMs.



$$P(L, D, A, E, S) = \frac{1}{Z} F_1(L) F_2(A, E) F_3(D, E, S)$$

Figure A.7: An example Markov network (undirected PGM)

Appendix B

The *Courses* Dataset

This chapter contains the specification of the *Courses* dataset, which is used for the experiments of Chapter 6.

The *Courses* dataset contains 17 single-table schemas as follows. Notice that the primary keys are underlined, and F_i is the set of functional dependencies of schema S_i, where the functional dependencies that are related to the primary keys are omitted.

$S_1(\underline{semester, course}, title, instructor, time, room)$

$F_1 = \{course \rightarrow title\}$

$S_2 = (\underline{semester, course\#}, call\#, name, instructor, days, time, room)$

$F_2 = \{course\# \rightarrow name\}$

$S_3 = (\underline{semester, course, sec\#}, code\#, room\#, time, days, units, instructor)$

$F_3 = \{code\# \rightarrow \{room\#, time, days, units, instructor\}\}$

$S_4 = (\underline{semester, course, crn}, time, day, instructor, room, units, title)$

$F_4 = \{course \rightarrow \{units, title\}\}$

$S_5 = (\underline{semester, course\#}, course, sched\#, days, time, location, instructor)$

$F_5 = \{course\# \rightarrow course\}$

$S_6 = (\underline{term, coursesubject, coursenumber, sectionID}, R, MeetingType, section, days, time, building, room, instructor, seats\text{-}available, seats\text{-}limit, coursename)$

$F_6 = \{\{coursesubject, coursenumber\} \rightarrow coursename\}$

$S_7 = (\underline{semester, year, course\text{-}abbr\text{-}num}, sec\text{-}num, avl, enrl\text{-}cnt, type, course\text{-}title, hr\text{-}cr, time\text{-}begin, time\text{-}end, days, room, building, special\text{-}enrollment, instructor)$

$F_7 = \{course\text{-}abbr\text{-}num \rightarrow \{type, course\text{-}title, hr\text{-}cr\}\}$

$S_8 = (\underline{semester, course\#}, ug, coursename, teacher, day, hour, place, comment)$

$F_8 = \{course\# \rightarrow coursename\}$

$S_9 = (\underline{term, course}, subject, instructor, time, room, units, grade)$

$F_9 = \{course \rightarrow subject\}$

$S_{10} = (\underline{semester, course}, title, units, lec/sec, days, begin, end, bldg, room, instructors)$

$F_{10} = \{course \rightarrow title\}$

$S_{11} = (\underline{semester, dept, course\#}, course, instructor, day, time, room)$

$F_{11} = \{\{dept, course\#\} \rightarrow course\}$

$S_{12} = (\underline{semester, code, sec}, course, time, days, cr, room, instructor)$

$F_{12} = \{code \rightarrow course\}$

$S_{13} = (\underline{semester, coursecode, class}, coursename, remark, instructor, time, building, roomnumber, requiredorelective, numberofstudents)$

$F_{13} = \{coursecode \rightarrow coursename\}$

$S_{14} = (\underline{semester, course, sec}, coursename, days, time, cr, instructor)$

$F_{14} = \{course \rightarrow coursename\}$

$$S_{15} = (\underline{semester, course\#, section\#}, coursename, cal\#, days, times, room, status, max, now, instructor, credits)$$

$$F_{15} = \{course\# \rightarrow coursename\}$$

$$S_{16} = (\underline{semester, coursenumber}, coursetitle, cr, days, time, bldg, room, insr, ref)$$

$$F_{16} = \{coursenumber \rightarrow coursetitle\}$$

$$S_{17} = (\underline{term, class\#}, title, location, day, times, instructors, status)$$

$$F_{17} = \{class\# \rightarrow title\}$$

Appendix C

Author's publications

- N. Ayat, R. Akbarinia, H. Afsarmanesh, and P. Valduriez. Entity resolution for distributed probabilistic data. *Distributed and Parallel Databases Journal*, 31(4): 509-542, 2013.
- N. Ayat, H. Afsarmanesh, R. Akbarinia, and P. Valduriez. Entity resolution for probabilistic data. *Information Sciences Journal*, 277: 492-511, 2014.
- N. Ayat, H. Afsarmanesh, R. Akbarinia, and P. Valduriez. Entity resolution for uncertain data using entropy reduction. Submitted to *Computer Journal* (under review), 2013.
- N. Ayat, H. Afsarmanesh, R. Akbarinia, and P. Valduriez. An uncertain data integration system. In *ODBASE*, pages 825-842, 2012.
- N. Ayat, H. Afsarmanesh, R. Akbarinia, and P. Valduriez. Entity resolution for uncertain data. In *BDA*, pages 135-154, 2012.
- N. Ayat, H. Afsarmanesh, R. Akbarinia, and P. Valduriez. Pay-as-you-go data integration using functional dependencies. In *CD-ARES*, pages 375-389, 2012.

Bibliography

[1] BayesStore. http://www.eecs.berkeley.edu/Research/Projects/Data/102060.html. Accessed: 2013.

[2] BRITE. http://www.cs.bu.edu/brite. Accessed: 2013.

[3] MayBMS. http://www.cs.cornell.edu/database-OLD/maybms. Accessed: 2013.

[4] ORION. http://orion.cs.purdue.edu. Accessed: 2013.

[5] PeerSim. http://peersim.sourceforge.net. Accessed: 2013.

[6] Pittsburgh Pattern Recognition. http://www.pittpatt.com. Accessed: 2013.

[7] PrDB. http://www.cs.umd.edu/~amol/PrDB. Accessed: 2013.

[8] Trio. http://infolab.stanford.edu/trio. Accessed: 2013.

[9] S. Abiteboul, P. C. Kanellakis, and G. Grahne. On the representation and querying of sets of possible worlds. *Theor. Comput. Sci.*, 78(1):158–187, 1991.

[10] 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 *VLDB*, pages 1151–1154, 2006.

[11] R. Akbarinia, P. Valduriez, and G. Verger. Efficient evaluation of sum queries over probabilistic data. *IEEE Trans. Knowl. Data Eng.*, 25(4):764–775, 2013.

[12] P. Andritsos, A. Fuxman, and R. J. Miller. Clean answers over dirty databases: A probabilistic approach. In *ICDE*, page 30, 2006.

[13] L. Antova, C. Koch, and D. Olteanu. World-set decompositions: Expressiveness and efficient algorithms. In *ICDT*, pages 194–208, 2007.

[14] A. Arasu, M. Götz, and R. Kaushik. On active learning of record matching packages. In *SIGMOD*, pages 783–794, 2010.

[15] A. Arasu, C. Ré, and D. Suciu. Large-scale deduplication with constraints using dedupalog. In *ICDE*, pages 952–963, 2009.

[16] N. Ayat, H. Afsarmanesh, R. Akbarinia, and P. Valduriez. Entity resolution for distributed uncertain data. In *DBDBD*, 2012.

[17] N. Ayat, H. Afsarmanesh, R. Akbarinia, and P. Valduriez. Entity resolution for uncertain data. In *BDA*, pages 135–154, 2012.

[18] N. Ayat, H. Afsarmanesh, R. Akbarinia, and P. Valduriez. Pay-as-you-go data integration using functional dependencies. In *CD-ARES*, pages 375–389, 2012.

[19] N. Ayat, H. Afsarmanesh, R. Akbarinia, and P. Valduriez. An uncertain data integration system. In *ODBASE*, pages 825–842, 2012.

[20] N. Ayat, H. Afsarmanesh, R. Akbarinia, and P. Valduriez. Entity resolution for uncertain data using entropy reduction. In *under review*, 2013.

[21] N. Ayat, H. Afsarmanesh, R. Akbarinia, and P. Valduriez. Entity resolution for probabilistic data. *Information Sciences*, 277(1):492–511, 2014.

[22] N. Ayat, R. Akbarinia, H. Afsarmanesh, and P. Valduriez. Entity resolution for distributed probabilistic data. *Distributed and Parallel Databases*, 31(4):509–542, 2013.

[23] R. Baxter, P. Christen, and T. Churches. A comparison of fast blocking methods for record linkage. In *ACM SIGKDD*, volume 3, pages 25–27, 2003.

[24] K. Bellare, S. Iyengar, A. G. Parameswaran, and V. Rastogi. Active sampling for entity matching. In *KDD*, pages 1131–1139, 2012.

[25] O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S. E. Whang, and J. Widom. Swoosh: a generic approach to entity resolution. *VLDB J.*, 18(1):255–276, 2009.

[26] A. Bhattacharjee and H. M. Jamil. Ontomatch: A monotonically improving schema matching system for autonomous data integration. In *IRI*, pages 318–323, 2009.

[27] I. Bhattacharya and L. Getoor. Collective entity resolution in relational data. *IEEE Data Eng. Bull.*, 29(2):4–12, 2006.

[28] I. Bhattacharya and L. Getoor. A latent dirichlet model for unsupervised entity resolution. In *SDM*, 2006.

[29] M. Bilenko, B. Kamath, and R. J. Mooney. Adaptive blocking: Learning to scale up record linkage. In *ICDM*, pages 87–96, 2006.

[30] M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *KDD*, pages 39–48, 2003.

[31] 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):16–23, 2003.

[32] J. Biskup and D. W. Embley. Extracting information from heterogeneous information sources using ontologically specified target views. *Inf. Syst.*, 28(3):169–212, 2003.

[33] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[34] C. Brew, D. McKelvie, et al. Word-pair extraction for lexicography. In *NeM-LaP*, pages 45–55, 1996.

[35] B. Buckles, F. Petry, and H. Sachar. Design of similarity-based relational databases. In *Fuzzy logic in Knowledge Engineering*, pages 3–7, 1986.

[36] B. P. Buckles and F. E. Petry. A fuzzy representation of data for relational databases. *Fuzzy sets and systems*, 7(3):213–226, 1982.

[37] B. P. Buckles and F. E. Petry. Uncertainty models in information and database systems. *Journal of Information Science*, 11(2):77–87, 1985.

[38] S. Chaudhuri, V. Ganti, and R. Motwani. Robust identification of fuzzy duplicates. In *ICDE*, pages 865–876, 2005.

[39] M. A. Cheema, X. Lin, W. Wang, W. Zhang, and J. Pei. Probabilistic reverse nearest neighbor queries on uncertain data. *IEEE Trans. Knowl. Data Eng.*, 22(4):550–564, 2010.

[40] P. Cheeseman and J. Stutz. Bayesian classification (autoclass): Theory and results. In *Advances in Knowledge Discovery and Data Mining*, pages 153–180. 1996.

[41] Z. Chen, D. V. Kalashnikov, and S. Mehrotra. Exploiting context analysis for combining multiple entity resolution systems. In *SIGMOD*, pages 207–218, 2009.

[42] R. Cheng, J. Chen, and X. Xie. Cleaning uncertain data with quality guarantees. *PVLDB*, 1(1):722–735, 2008.

[43] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *SIGMOD*, pages 551–562, 2003.

[44] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Querying imprecise data in moving object environments. *TKDE*, 16(9):1112–1127, 2004.

[45] P. Christen. Automatic record linkage using seeded nearest neighbour and support vector machine classification. In *KDD*, pages 151–159, 2008.

[46] M. Cochinwala, V. Kurien, G. Lalk, and D. Shasha. Efficient data reconciliation. *Inf. Sci.*, 137(1-4):1–15, 2001.

[47] E. F. Codd. Understanding relations (installment #7). *FDT - Bulletin of ACM SIGMOD*, 7(3):23–28, 1975.

[48] E. F. Codd. Extending the database relational model to capture more meaning. *ACM Trans. Database Syst.*, 4(4):397–434, 1979.

[49] W. W. Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *SIGMOD*.

[50] W. W. Cohen, P. D. Ravikumar, and S. E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *IIWeb*, pages 73–78, 2003.

[51] D. A. Cohn, L. E. Atlas, and R. E. Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, 1994.

[52] G. Cormode, F. Li, and K. Yi. Semantics of ranking queries for probabilistic data and expected ranks. In *ICDE*, pages 305–316, 2009.

[53] I. Davidson and S. S. Ravi. Using instance-level constraints in agglomerative hierarchical clustering: theoretical and empirical results. *Data Min. Knowl. Discov.*, 18(2):257–282, 2009.

[54] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–38, 1977.

[55] A. Deshpande, L. Getoor, and P. Sen. Graphical models for uncertain data. In C. Aggarwal, editor, *Managing and Mining Uncertain Data*. Springer, 2009.

[56] P. Diaconis and R. L. Graham. Spearman's footrule as a measure of disarray. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 262–268, 1977.

[57] P. Domingos. Multi-relational record linkage. In *KDD Workshop on Multi-Relational Data Mining*, 2004.

[58] X. Dong, A. Y. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *SIGMOD*, pages 85–96, 2005.

[59] X. L. Dong, A. Y. Halevy, and C. Yu. Data integration with uncertainty. *VLDB J.*, 18(2):469–500, 2009.

[60] U. Draisbach and F. Naumann. A generalization of blocking and windowing algorithms for duplicate detection. In *ICDKE*, pages 18–24, 2011.

[61] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.*, 19(1):1–16, 2007.

[62] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969.

[63] N. Fuhr and T. Rölleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Trans. Inf. Syst.*, 15(1):32–66, 1997.

[64] H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C.-A. Saita. Declarative data cleaning: Language, model, and algorithms. In *VLDB*, pages 371–380, 2001.

[65] G. Grahne. Dependency satisfaction in databases with incomplete information. In *VLDB*, pages 37–45, 1984.

[66] J. Grant. Null values in a relational data base. *Inf. Process. Lett.*, 6(5):156–157, 1977.

[67] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, L. Pietarinen, and D. Srivastava. Using q-grams in a dbms for approximate string processing. *IEEE Data Eng. Bull.*, 24(4):28–34, 2001.

[68] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *VLDB*, pages 491–500, 2001.

[69] L. Gravano, P. G. Ipeirotis, N. Koudas, and D. Srivastava. Text joins in an rdbms for web data integration. In *WWW*, pages 90–101, 2003.

[70] L. Gu and R. A. Baxter. Adaptive filtering for efficient record linkage. In *SDM*, 2004.

[71] S. Guha, N. Koudas, A. Marathe, and D. Srivastava. Merging the results of approximate match operations. In *VLDB*, pages 636–647, 2004.

[72] R. Gupta and S. Sarawagi. Creating probabilistic databases from information extraction models. In *VLDB*, pages 965–976, 2006.

[73] R. Gupta and S. Sarawagi. Answering table augmentation queries from unstructured lists on the web. *PVLDB*, 2(1):289–300, 2009.

[74] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *SIGKDD Explorations*, 11(1):10–18, 2009.

[75] O. Hassanzadeh and R. J. Miller. Creating probabilistic databases from duplicated data. *VLDB J.*, 18(5):1141–1166, 2009.

[76] M. A. Hernández and S. J. Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Min. Knowl. Discov.*, 2(1):9–37, 1998.

[77] Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen. Tane: An efficient algorithm for discovering functional and approximate dependencies. *Comput. J.*, 42(2):100–111, 1999.

[78] T. Imielinski and W. L. Jr. Incomplete information in relational databases. *J. ACM*, 31(4):761–791, 1984.

[79] T. Inagaki. Interdependence between safety-control policy and multiple-sensor schemes via dempster-shafer theory. *IEEE Trans. on Reliability*, 40(2):182–188, 1991.

[80] M. A. Jaro. Unimatch: A record linkage system: User's manual. Technical report, US Bureau of the Census.

[81] N. Koudas, A. Marathe, and D. Srivastava. Flexible string matching against large databases in practice. In *VLDB*, pages 1078–1086, 2004.

[82] H.-P. Kriegel, P. Kunath, and M. Renz. Probabilistic nearest-neighbor query on uncertain objects. In *DASFAA*, pages 337–348, 2007.

[83] L. V. S. Lakshmanan, N. Leone, R. B. Ross, and V. S. Subrahmanian. Probview: A flexible probabilistic database system. *ACM Trans. Database Syst.*, 22(3):419–469, 1997.

[84] J. A. Larson, S. B. Navathe, and R. Elmasri. A theory of attribute equivalence in databases with application to schema integration. *IEEE Trans. Software Eng.*, 15(4):449–463, 1989.

[85] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10.

[86] F. Li, K. Yi, and J. Jestes. Ranking distributed probabilistic data. In *SIGMOD*, pages 361–374, 2009.

[87] F.-F. Li and P. Perona. A bayesian hierarchical model for learning natural scene categories. In *CVPR (2)*, pages 524–531, 2005.

[88] E.-P. Lim, J. Srivastava, S. Prabhakar, and J. Richardson. Entity identification in database integration. *Inf. Sci.*, 89(1):1–38, 1996.

[89] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110, 2004.

[90] C. Manning, P. Raghavan, and H. Schutze. *Introduction to information retrieval*, volume 1. Cambridge University Press, 2008.

[91] A. Marcus, E. Wu, D. R. Karger, S. Madden, and R. C. Miller. Human-powered sorts and joins. *PVLDB*, 5(1):13–24, 2011.

[92] A. McCallum, K. Nigam, and L. H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *KDD*, pages 169–178, 2000.

[93] A. McCallum and B. Wellner. Conditional models of identity uncertainty with application to noun coreference. In *NIPS*, 2004.

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

[95] M. Michelson and C. A. Knoblock. Learning blocking schemes for record linkage. In *AAAI*, pages 440–445, 2006.

[96] A. E. Monge and C. Elkan. The field matching problem: Algorithms and applications. In *KDD*, pages 267–270, 1996.

[97] A. E. Monge and C. Elkan. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *DMKD*, pages 0–, 1997.

[98] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Molecular Biology*, 48(3):443–453, 1970.

[99] H. B. Newcombe and J. M. Kennedy. Record linkage: making maximum use of the discriminating power of identifying information. *Commun. ACM*, 5(11):563–566, Nov. 1962.

[100] M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems, 3rd Edition*. Springer, 2011.

[101] L. Palopoli, G. Terracina, and D. Ursino. Dike: a system supporting the semi-automatic construction of cooperative information systems from heterogeneous databases. *Softw., Pract. Exper.*, 33(9):847–884, 2003.

[102] F. Panse and N. Ritter. Tuple merging in probabilistic databases. In *MUD*, pages 113–127, 2010.

[103] F. Panse, M. van Keulen, A. de Keijzer, and N. Ritter. Duplicate detection in probabilistic data. In *ICDE Workshops*, pages 179–182, 2010.

[104] L. Peng, Y. Diao, and A. Liu. Optimizing probabilistic query processing on continuous uncertain data. *PVLDB*, 4(11):1169–1180, 2011.

[105] H. Prade. Lipski's approach to incomplete information databases restated and generalized in the setting of zadeh's possibility theory. *Inf. Syst.*, 9(1):27–42, 1984.

[106] H. Prade and C. Testemale. Generalizing database relational algebra for the treatment of incomplete/uncertain information and vague queries. *Inf. Sci.*, 34(2):115–143, 1984.

[107] W. C. Purdy. A logic for natural language. *Notre Dame Journal of Formal Logic*, 32(3):409–425, 1991.

[108] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350, 2001.

[109] K. V. S. V. N. Raju and A. K. Majumdar. Fuzzy functional dependencies and lossless join decomposition of fuzzy relational database systems. *ACM Trans. Database Syst.*, 13(2):129–166, 1988.

[110] V. Rastogi, N. N. Dalvi, and M. N. Garofalakis. Large-scale collective entity matching. *PVLDB*, 4(4):208–218, 2011.

[111] P. D. Ravikumar and W. W. Cohen. A hierarchical graphical model for record linkage. In *UAI*, pages 454–461, 2004.

[112] C. Re, N. N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *ICDE*, pages 886–895, 2007.

[113] E. S. Ristad and P. N. Yianilos. Learning string-edit distance. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(5):522–532, 1998.

[114] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *KDD*, pages 269–278, 2002.

[115] A. D. Sarma, X. Dong, and A. Y. Halevy. Bootstrapping pay-as-you-go data integration systems. In *SIGMOD*, pages 861–874, 2008.

[116] A. D. Sarma, A. Jain, A. Machanavajjhala, and P. Bohannon. An automatic blocking mechanism for large-scale de-duplication tasks. In *CIKM*, pages 1055–1064, 2012.

[117] R. A. Schmidt. Relational grammars for knowledge representation. In M. Böttner and W. Thümmel, editors, *Variable-Free Semantics*, volume 3 of *Artikulation und Sprache*, pages 162–180. secolo Verlag, Osnabrück, Germany, 2000.

[118] G. Shafer. *A mathematical theory of evidence*. Princeton University Press, 1976.

[119] C. E. Shannon. A mathematical theory of communication. *Mobile Computing and Communications Review*, 5(1):3–55, 2001.

[120] T. Smith and M. Waterman. Identification of common molecular subsequences. *J. Molecular Biology*, 147:195–197, 1981.

[121] M. A. Soliman, I. F. Ilyas, and K. C.-C. Chang. Top-k query processing in uncertain databases. In *ICDE*, pages 896–905, 2007.

[122] D. Suciu, A. Connolly, and B. Howe. Embracing uncertainty in large-scale computational astrophysics. In *MUD*, pages 63–77, 2009.

[123] E. Sutinen and J. Tarhio. On using q-gram locations in approximate string matching. In *ESA*, pages 327–340, 1995.

[124] S. Tejada, C. A. Knoblock, and S. Minton. Learning object identification rules for information integration. *Inf. Syst.*, 26(8):607–633, 2001.

[125] G. Trajcevski, R. Tamassia, H. Ding, P. Scheuermann, and I. F. Cruz. Continuous probabilistic nearest-neighbor queries for uncertain trajectories. In *EDBT*, pages 874–885, 2009.

[126] E. Ukkonen. Approximate string matching with q-grams and maximal matches. *Theor. Comput. Sci.*, 92(1):191–211, 1992.

[127] J. R. Ullmann. A binary n-gram technique for automatic correction of substitution, deletion, insertion and reversal errors in words. *Comput. J.*, 20(2):141–147, 1977.

[128] M. Umano. Freedom-0: a fuzzy database system. In *Fuzzy information and decision processes*, pages 339–347. North-Holland, Amsterdam, 1982.

[129] M. Umano. Retrieval from fuzzy database by fuzzy relational algebra. In *Fuzzy Information, Knowledge Representation and Decision Analysis*, pages 1–6, 1983.

[130] O. Unal and H. Afsarmanesh. Semi-automated schema integration with sasmint. *Knowl. Inf. Syst.*, 23(1):99–128, 2010.

[131] V. S. Verykios, A. K. Elmagarmid, and E. N. Houstis. Automating the approximate record-matching process. *Inf. Sci.*, 126(1-4):83–98, 2000.

[132] D. Z. Wang, X. L. Dong, A. D. Sarma, M. J. Franklin, and A. Y. Halevy. Functional dependency generation and applications in pay-as-you-go data integration systems. In *WebDB*, 2009.

[133] J. Wang, T. Kraska, M. J. Franklin, and J. Feng. Crowder: Crowdsourcing entity resolution. *PVLDB*, 5(11):1483–1494, 2012.

[134] Y. R. Wang and S. E. Madnick. The inter-database instance identification problem in integrating autonomous systems. In *ICDE*, pages 46–55, 1989.

[135] M. S. Waterman, T. F. Smith, and W. A. Beyer. Some biological sequence metrics. *Advances in Mathematics*, 20(3):367–387, 1976.

[136] S. E. Whang, D. Menestrina, G. Koutrika, M. Theobald, and H. Garcia-Molina. Entity resolution with iterative blocking. In *SIGMOD*, pages 219–232, 2009.

[137] W. E. Winkler. Approximate string comparator search strategies for very large administrative lists. Technical report, Statistical Research Division, US Bureau of the Census.

[138] W. E. Winkler. The state of record linkage and current research problems. Technical report, Statistical Research Division, US Census Bureau.

[139] W. E. Winkler and Y. Thibaudeau. An application of the fellegi-sunter model of record linkage to the 1990 us decennial census. Technical report statistical research report series RR91/09, US Bureau of the Census.

[140] M. Ye, X. Liu, W.-C. Lee, and D. L. Lee. Probabilistic top-k query processing in distributed sensor networks. In *ICDE*, pages 585–588, 2010.

[141] K. Yi, F. Li, G. Kollios, and D. Srivastava. Efficient processing of top-k queries in uncertain databases with x-relations. *IEEE Trans. Knowl. Data Eng.*, 20(12):1669–1682, 2008.

[142] K. Yi, F. Li, G. Kollios, and D. Srivastava. Efficient processing of top-k queries in uncertain databases with x-relations. *TKDE*, 20(12):1669–1682, 2008.

[143] S. M. Yuen, Y. Tao, X. Xiao, J. Pei, and D. Zhang. Superseding nearest neighbor search on uncertain spatial databases. *TKDE*, 22(7):1041–1055, 2010.

[144] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965.

[145] Q. Zhang, F. Li, and K. Yi. Finding frequent items in probabilistic data. In *SIGMOD*, pages 819–832, 2008.

[146] S. K. Zhou, V. Krüger, and R. Chellappa. Probabilistic recognition of human faces from video. *Computer Vision and Image Understanding*, 91(1-2):214–245, 2003.

Abstract

Entity resolution (ER) is the problem of identifying duplicate tuples, which are the tuples that represent the same real-world entity. There are many real-life applications in which the ER problem arises. These applications range from *news* aggregation websites, identifying the news that cover the same story, in order to avoid presenting one story several times to the user, to the integration of two companies' customer databases in the case of a merger, where identifying the tuples that refer to the same *customer* is crucial.

Due to its diverse applications, the ER problem has been formulated in different ways in the literature. The two main ER's related problem formulations include: 1) *identity resolution*, and 2) *deduplication*. In identity resolution, the aim is to find duplicate(s) of a given tuple in a given database, while in deduplication, the aim is to find groups of duplicate tuples in a given database, and merge them in order to increase the quality of the database itself.

The ER problem is however not limited to deterministic (ordinary) databases, rather it also arises in applications that deal with probabilistic databases, i.e. databases in which each tuple or attribute value is associated with a probability value to, for instance, indicate its confidence level. In this thesis, we study the ER problem in probabilistic databases. More specifically, we address five challenges described in the following paragraphs.

The first challenge is that in contrast to deterministic data, in probabilistic data, the semantics of identity resolution problem is not clear. In identity resolution over *deterministic* data, the aim is to match the *most similar* tuple in the database to a given tuple. However the aim is not so clear when matching probabilistic entities, since we have to deal with the two concepts of the *most similar* and the *most probable*, at the same time.

Efficient dealing with the identity resolution problem in probabilistic data is the second challenge that we address in this thesis. In order to define the semantics of the identity resolution problem over probabilistic data, we use the *possible worlds* semantics of uncertain data, treating a probabilistic database as

the probability distribution over a set of *deterministic* database instances, each of which is called a *possible world*. Each possible world thus, is a deterministic database which occurs with certain probability. The number of possible worlds of a probabilistic database might easily be exponential, which thus makes the naïve computation of the defined semantics impractical.

In many applications that the identity resolution problem arises, probabilistic data is distributed among a number of nodes. Dealing with the identity resolution problem in distributed probabilistic data, while reducing the amount of exchanged data among nodes, is quite challenging. Efficient dealing with the identity resolution problem over probabilistic data in *distributed systems* is the third challenge that we address in this thesis.

The fourth challenge is raised by considering the *deduplication* problem in probabilistic data. Similar to *deterministic* data, the aim of deduplication in probabilistic data is to improve the quality of the database. However, we observe that deduplication does not necessarily improve the quality of the probabilistic database. Therefore, guaranteeing the quality improvement of probabilistic databases by a deduplication approach is another challenge that we address in this thesis.

In many applications where the ER problem arises, data resides in a number of heterogenous data sources. In such applications, matching the heterogeneous schemas of data sources, to which we refer as *schema matching*, is an inevitable step in dealing with the ER problem. On the other hand, dealing with the schema matching problem requires human knowledge about the context, which is in contrast to the full automated resolution setting, which we propose for applications in which the ER problem arises. Thus, effective dealing with the schema matching problem in a fully automated setting is the fifth challenge addressed in this thesis.

The thesis is structured as follows.

In Chapter 1, we elaborate on the motivation of this work, and present the research questions and contributions of this research.

Chapter 2 provides preliminary definitions and concepts that are used throughout the thesis. We first present the uncertain data models, and then review the related work on entity resolution area.

In Chapter 3, we deal with the identity resolution problem over probabilistic data. We adapt the possible worlds semantics of uncertain data to define the semantics of identity resolution problem in probabilistic data. Our approach for computing the defined semantics depends of the similarity function, which is used for computing the similarity between tuples. We differentiate 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 the context-sensitive similarity functions. We deal with the problem of high response time of existing context-sensitive similarity functions, which makes them very inefficient for the Monte Carlo algorithm, by

proposing a new efficient context-sensitive similarity function that is very appropriate for the Monte Carlo algorithm. We further speed up our proposed Monte Carlo algorithm by parallelizing it using the MapReduce framework.

Chapter 4 deals with the identity resolution problem over distributed probabilistic data. We propose a fully distributed algorithm for computing the semantics of the identity resolution problem, as defined in Chapter 3, in a distributed system. Our algorithm prunes data at local nodes, which thus results in significant reduction in bandwidth usage and the response time compared to the baseline approaches. Moreover, it requires no global information, and does not depend on the existence of certain nodes.

In Chapter 5, we deal with deduplication problem over probabilistic data with the aim of improving the quality of probabilistic data. We use the amount of uncertainty, i.e. entropy, of the probabilistic database as a quality metric and propose an efficient technique for computing it. We then propose a merge function for merging probabilistic duplicate tuples. Further, we propose an efficient algorithm that uses our proposed merge function and produces a cleaned database with (near) minimum entropy. This leads to a significant improvement in the results of the queries which are posed over the database.

Chapter 6 aims at building a data integration system in a fully automated setting. The main problem that is dealt with in this chapter is the schema matching problem, which arises in many applications that need to deal with the entity resolution problem. We propose an algorithm that takes advantage of the background knowledge implied in *functional dependencies* for finding attribute correlations and using it for matching the source schemas. and generating the mediated schema. Our algorithm is built on a probabilistic data model in order to model the uncertainty in data integration systems.

Finally, Chapter 7 concludes and gives some research directions for future work.

Samenvatting

Het identificeren van gedupliceerde tuples in data wordt ook wel het *Entity Resolution* (ER) probleem genoemd. Deze gedupliceerde tuples zijn tuples die hetzelfde concept representeren. Dit komt voor bij verschillende toepassingen zoals nieuwsaggregatie websites waar nieuwsberichten die over hetzelfde onderwerp gaan worden geïdentificeerd. Hiermee wordt voorkomen dat meerdere berichten over hetzelfde onderwerp aan de gebruiker worden gepresenteerd. Een andere toepassing is de migratie van klantgegevens bij een fusie waar het identificeren van tuples die naar dezelfde klant verwijzen cruciaal is.

Vanwege de diversiteit in toepassingen wordt het ER probleem in de literatuur op verschillende manieren geformuleerd. De twee meest voorkomende formuleringen zijn *identity resolution* en *deduplication*. Het doel in *identity resolution* is om duplicaten van een gegeven tuple in een gegeven database te vinden. Het doel van *deduplication* is om gedupliceerde tuples in een gegeven database te vinden en ze samen te voegen om zodoende de kwaliteit van de database te verbeteren.

Het ER probleem beperkt zich echter niet tot (normale) deterministische databases. Het komt namelijk ook voor in toepassingen met probabilistische databases. Dit zijn databases waar elk tuple of attribuut wordt geassocieerd met een kans die bijvoorbeeld een indicatie is voor de betrouwbaarheid. In dit proefschrift wordt het ER probleem in probabilistische databases bestudeerd. Hierbij komen de volgende vijf uitdagingen aan bod.

De eerste uitdaging is dat in tegenstelling tot deterministische data, bij probabilistische data de semantiek van het *identity resolution* probleem niet duidelijk is. Bij *identity resolution* op deterministische data is het doel om het meest overeenkomende tuple in de database aan een tuple te koppelen. Het doel bij het koppelen van probabilistische data is niet zo duidelijk omdat naast de notie van het meest overeenkomende tuple ook de notie van het meest waarschijnlijke tuple bestaat.

Het op een efficiënte manier omgaan met *identity resolution* in probabilistische data is de tweede uitdaging die wordt beschreven in dit proefschrift. Om de

semantiek van *identity resolution* op probabilistische data te definiëren maken we gebruik van *possible worlds* (mogelijke werelden) semantiek van onzekere data. Hiermee beschouwen we een probabilistische database als een kansverdeling over een verzameling van deterministische databases. Elk van deze databases is een zogenaamde mogelijke wereld. Daarmee is elke mogelijke wereld een deterministische database die bestaat met een bepaalde kans. Het aantal mogelijke werelden van een probabilistische database kan exponentieel zijn en dit maakt naïeve berekening van de gedefinieerde semantiek in de praktijk niet haalbaar.

In veel toepassingen waar het probleem van *identity resolution* zich voordoet is de probabilistische data verspreid over een aantal nodes. Het op een efficiënte manier omgaan met gedistribueerde probabilistische data bij het toepassen van *identity resolution* is de derde uitdaging die in dit proefschrift wordt beschreven.

De vierde uitdaging is *deduplication* in probabilistische data. Net als bij deterministische data, is het doel van *deduplication* bij probabilistische data om de kwaliteit van de database te verbeteren. We stellen echter vast dat het niet vanzelfsprekend is dat de kwaliteit van de probabilistische database ook daadwerkelijk verbetert bij het toepassen van *deduplication*. Daarom is het garanderen van een kwaliteitsverbetering van een probabilistische database bij het toepassen van *deduplication* een van de uitdagingen die worden beschreven in dit proefschrift.

In veel toepassingen van het ER probleem bevindt de data zich in een aantal heterogene databronnen. *Schema matching* wordt gebruikt om de schema's van heterogene databronnen op elkaar aan te laten sluiten. In deze toepassingen van het ER probleem is *schema matching* een onvermijdelijke stap in het oplossen van het ER probleem. Het omgaan met *schema matching* vereist menselijke kennis over de context van de data. Dit is in tegenstelling tot de volledig geautomatiseerde ER die we voorstellen bij toepassingen waar het ER probleem zich voordoet. Het op een effectieve manier omgaan met *schema matching* op een volledig geautomatiseerde manier is daarom de vijfde uitdaging die in dit proefschrift wordt beschreven.

De structuur van dit proefschrift is als volgt.

In hoofdstuk 1 beschrijven we de motivatie achter dit onderzoek, de onderzoeksvragen die worden beantwoord in dit proefschrift en de bijdragen van dit onderzoek.

Hoofdstuk 2 bevat de voorlopige definities en concepten die in het gehele proefschrift worden gebruikt. We presenteren de onzekere datamodellen en bespreken de literatuur op het gebied van ER.

In hoofdstuk 3 beschrijven we ER in de context van probabilistische data. Hierbij wordt de *possible worlds* semantiek toegepast op onzekere data om op die manier de semantiek van ER in probabilistische data te definiëren. Deze aanpak voor het berekenen van de gedefinieerde semantiek is afhankelijk van een *similarity* functie die wordt gebruikt om de mate van overeenkomst tussen twee tuples te berekenen. Hierbij maken we onderscheid tussen twee klassen van *similarity*

functies, de contextvrije en de contextgevoelige functies. Vervolgens introduceren we een zogenaamd PTIME algoritme voor contextvrije *similarity* functies en we gebruiken Monte Carlo benadering voor de contextgevoelige *similarity* functies. Bestaande contextgevoelige *similarity* functies hebben hoge response tijden en dit maakt deze functies erg inefficiënt bij het toepassen van het Monte Carlo algoritme. In dit hoofdstuk introduceren we een nieuwe en efficiënte contextgevoelige *similarity* functie die wel geschikt is voor het Monte Carlo algoritme. Een verdere optimalisatie wordt bereikt door het paralelliseren van het algoritme door middel van het MapReduce raamwerk.

Hoofdstuk 4 beschrijft het *identity resolution* probleem in gedistribueerde probabilistische data. We introduceren een volledig gedistribueerd algoritme om de semantiek van het *identity resolution* probleem, zoals beschreven in hoofdstuk 3, te kunnen berekenen. Ons algoritme vereenvoudigt de data op elke node, waarmee een significante vermindering van de benodigde brandbreedte en response tijden wordt gerealiseerd in vergelijking met de standaard aanpak. Een ander voordeel van dit algoritme is dat het geen globale informatie nodig heeft en niet afhankelijk is van het bestaan van bepaalde nodes.

Hoofdstuk 5 gaat over het *deduplication* probleem in probabilistische data met als doel het verbeteren van de kwaliteit van deze data. We gebruiken de hoeveelheid onzekerheid, oftewel de entropie, van de probabilistische database als een kwaliteitsmetriek. Vervolgens introduceren we een efficiënte manier om deze metriek te berekenen. Ook introduceren we een functie om gedupliceerde probabilistische tuples samen te voegen. Deze functie wordt gebruikt om een efficient algoritme te ontwerpen dat een geschoonde database met een zo klein mogelijke entropie produceert. Dit leidt tot een significante verbetering in de resultaten van de zoekopdrachten op de database.

Het doel van hoofdstuk 6 is om een volledig geautomatiseerd data integratie systeem te bouwen. Het grootste probleem dat in dit hoofdstuk wordt aangepakt is het *schema matching* probleem. We introduceren een algoritme dat gebruik maakt van achtergrond kennis die impliciet aanwezig is in de functionele afhankelijkheden. We gebruiken deze kennis voor het vinden van correlaties en gebruiken dit voor het aansluiten van de bronschema's en het genereren van een zogenaamd *mediated* schema. Het algoritme is gebaseerd op een probabilistisch data model om zodoende de onzekerheid in data-integratie systemen te kunnen modelleren.

Als laatste worden in hoofdstuk 7 de conclusies beschreven en worden er mogelijkheden voor toekomstig onderzoek besproken.

Acknowledgments

A teacher affects eternity; he can never tell where his influence stops.
H. B. Adams

This work would not have been possible without the help, and support of many people to whom I wish to express my sincere gratitude.

First and foremost, I owe my deepest gratitude to my great supervisors, Hamideh Afsarmanesh, Patrick Valduriez, and Reza Akbarinia, for all their help in my journey towards being an independent researcher. I believe that our easily-formed collaboration is mostly attributed to your attitude towards doing quality research. I hope and think that we continue collaborating in the future.

Hamideh, I really enjoyed your support throughout these years. You gave me a lot of freedom in deciding my research direction, but also guided me when necessary. Your invaluable comments, and scientific insights helped me a lot in preparing this work.

Patrick, I really enjoyed your support, stress-free style of collaboration, and professional guidance. My work has greatly benefited from your constructive recommendations.

Reza, I learned a lot from you. I really enjoyed our technical discussions, and your invaluable insights throughout these years. This work would not have been possible without your help.

I would like to express my great appreciation to Farhad Arbab, for his invaluable advices in my first year of PhD studies.

I am very thankful to my committee members, Pieter Adriaans, Peter Sloot, Arno Siebes, Marian Bubak, and Frans Groen, for agreeing to be on my committee and critically reading my thesis.

I wish to thank Michel Mandjes, and Guido van 't Noordende, for their invaluable help.

I would like to thank Mattijs Ghijsen, for writing the Samenvatting section, and Miriam Ghijsen, for her help.

Words cannot express my deep gratitude to my father, Mohammad, and my mother, Belgheis. They have always supported me in my life, and offered me unconditional love. Their belief in me has always been my motivation through the hard times.

A big thanks to my brother, Yaser, and my sister, Saba, for being the persons on whom I always have been able to count, and their love.

I would like to express my gratitude to my father in-law, Mahmoud, and my mother in-law, Shahin, for their support, and love. Thanks to my brother and sisters in laws, Aydin, Somaye, Ooldooz, Aytak and little Arash, for making my life beautiful, and their warmth.

During research years, I have been fortunate to meet wonderful people in my professional, and personal life, which I could spend pages to name them, but singling out some of them would be unfair. Dear colleagues, and friends thank you all for welcoming me as a friend and helping to shape and develop the ideas in my thesis.

Last but not least, my thanks go to my lovely wife, Aylar, for the joyful sense of life that she gave me, and being with me all along.

SIKS Dissertation Series

1998-1 Johan van den Akker (CWI), DEGAS - An Active, Temporal Database of Autonomous Objects.

1998-2 Floris Wiesman (UM), Information Retrieval by Graphically Browsing Meta-Information.

1998-3 Ans Steuten (TUD), A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective.

1998-4 Dennis Breuker (UM), Memory versus Search in Games.

1998-5 E.W.Oskamp (RUL), Computerondersteuning bij Straftoemeting.

1999-1 Mark Sloof (VU), Physiology of Quality Change Modelling; Automated modelling of Quality Change of Agricultural Products.

1999-2 Rob Potharst (EUR), Classification using decision trees and neural nets.

1999-3 Don Beal (UM), The Nature of Minimax Search.

1999-4 Jacques Penders (UM), The practical Art of Moving Physical Objects.

1999-5 Aldo de Moor (KUB), Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems.

1999-6 Niek J.E. Wijngaards (VU), Re-design of compositional systems.

1999-7 David Spelt (UT), Verification support for object database design.

1999-8 Jacques H.J. Lenting (UM), Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation.

2000-1 Frank Niessink (VU), Perspectives on Improving Software Maintenance.

2000-2 Koen Holtman (TUE), Prototyping of CMS Storage Management.

2000-3 Carolien M.T. Metselaar (UVA), Sociaal-organisatorische gevolgen van kennistechnologie; een procesbenadering en actorperspectief.

2000-4 Geert de Haan (VU), ETAG, A Formal Model of Competence Knowledge for User Interface Design.

2000-5 Ruud van der Pol (UM), Knowledge-based Query Formulation in Information Retrieval.

2000-6 Rogier van Eijk (UU), Programming Languages for Agent Communication.

2000-7 Niels Peek (UU), Decision-theoretic Planning of Clinical Patient Management.

2000-8 Veerle Coup (EUR), Sensitivity Analyis of Decision-Theoretic Networks.

2000-9 Florian Waas (CWI), Principles of Probabilistic Query Optimization.

2000-10 Niels Nes (CWI), Image Database Management System Design Considerations, Algorithms and Architecture.

2000-11 Jonas Karlsson (CWI), Scalable Distributed Data Structures for Database Management.

2001-1 Silja Renooij (UU), Qualitative Approaches to Quantifying Probabilistic Networks.

2001-2 Koen Hindriks (UU), Agent Programming Languages: Programming with Mental Models.

2001-3 Maarten van Someren (UvA), Learning as problem solving.

2001-4 Evgueni Smirnov (UM), Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets.

2001-5 Jacco van Ossenbruggen (VU), Processing Structured Hypermedia: A Matter of Style.

2001-6 Martijn van Welie (VU), Task-based User Interface Design.

2001-7 Bastiaan Schonhage (VU), Diva: Architectural Perspectives on Information Visualization.

2001-8 Pascal van Eck (VU), A Compositional

Semantic Structure for Multi-Agent Systems Dynamics.

2001-9 Pieter Jan 't Hoen (RUL), Towards Distributed Development of Large Object-Oriented Models, Views of Packages as Classes.

2001-10 Maarten Sierhuis (UvA), Modeling and Simulating Work Practice BRAHMS: a multiagent modeling and simulation language for work practice analysis and design.

2001-11 Tom M. van Engers (VUA), Knowledge Management: The Role of Mental Models in Business Systems Design.

2002-01 Nico Lassing (VU), Architecture-Level Modifiability Analysis.

2002-02 Roelof van Zwol (UT), Modelling and searching web-based document collections.

2002-03 Henk Ernst Blok (UT), Database Optimization Aspects for Information Retrieval.

2002-04 Juan Roberto Castelo Valdueza (UU), The Discrete Acyclic Digraph Markov Model in Data Mining.

2002-05 Radu Serban (VU), The Private Cyberspace Modeling Electronic Environments inhabited by Privacy-concerned Agents.

2002-06 Laurens Mommers (UL), Applied legal epistemology; Building a knowledge-based ontology of the legal domain.

2002-07 Peter Boncz (CWI), Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications.

2002-08 Jaap Gordijn (VU), Value Based Requirements Engineering: Exploring Innovative E-Commerce Ideas.

2002-09 Willem-Jan van den Heuvel(KUB), Integrating Modern Business Applications with Objectified Legacy Systems.

2002-10 Brian Sheppard (UM), Towards Perfect Play of Scrabble.

2002-11 Wouter C.A. Wijngaards (VU), Agent Based Modelling of Dynamics: Biological and Organisational Applications.

2002-12 Albrecht Schmidt (Uva), Processing XML in Database Systems.

2002-13 Hongjing Wu (TUE), A Reference Architecture for Adaptive Hypermedia Applications.

2002-14 Wieke de Vries (UU), Agent Interaction: Abstract Approaches to Modelling, Programming and Verifying Multi-Agent Systems.

2002-15 Rik Eshuis (UT), Semantics and Verification of UML Activity Diagrams for Workflow Modelling.

2002-16 Pieter van Langen (VU), The Anatomy of Design: Foundations, Models and Applications.

2002-17 Stefan Manegold (UVA), Understanding, Modeling, and Improving Main-Memory Database Performance.

2003-01 Heiner Stuckenschmidt (VU), Ontology-Based Information Sharing in Weakly Structured Environments.

2003-02 Jan Broersen (VU), Modal Action Logics for Reasoning About Reactive Systems.

2003-03 Martijn Schuemie (TUD), Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy.

2003-04 Milan Petkovic (UT), Content-Based Video Retrieval Supported by Database Technology.

2003-05 Jos Lehmann (UVA), Causation in Artificial Intelligence and Law - A modelling approach.

2003-06 Boris van Schooten (UT), Development and specification of virtual environments.

2003-07 Machiel Jansen (UvA), Formal Explorations of Knowledge Intensive Tasks.

2003-08 Yongping Ran (UM), Repair Based Scheduling.

2003-09 Rens Kortmann (UM), The resolution of visually guided behaviour.

2003-10 Andreas Lincke (UvT), Electronic Business Negotiation: Some experimental studies on the interaction between medium, innovation context and culture.

2003-11 Simon Keizer (UT), Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks.

2003-12 Roeland Ordelman (UT), Dutch speech recognition in multimedia information retrieval.

2003-13 Jeroen Donkers (UM), Nosce Hostem - Searching with Opponent Models.

2003-14 Stijn Hoppenbrouwers (KUN), Freezing Language: Conceptualisation Processes across ICT-Supported Organisations.

2003-15 Mathijs de Weerdt (TUD), Plan Merging in Multi-Agent Systems.

2003-16 Menzo Windhouwer (CWI), Feature Grammar Systems - Incremental Maintenance of Indexes to Digital Media Warehouses.

2003-17 David Jansen (UT), Extensions of Statecharts with Probability, Time, and Stochastic Timing.

2003-18 Levente Kocsis (UM), Learning Search Decisions.

2004-01 Virginia Dignum (UU), A Model for Organizational Interaction: Based on Agents, Founded in Logic.

2004-02 Lai Xu (UvT), Monitoring Multi-party Contracts for E-business.

2004-03 Perry Groot (VU), A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving.

2004-04 Chris van Aart (UVA), Organizational Principles for Multi-Agent Architectures.

2004-05 Viara Popova (EUR), Knowledge discovery and monotonicity.

2004-06 Bart-Jan Hommes (TUD), The Evaluation

of Business Process Modeling Techniques.

2004-07 Elise Boltjes (UM), Voorbeeldig onderwijs; voorbeeldgestuurd onderwijs, een opstap naar abstract denken, vooral voor meisjes.

2004-08 Joop Verbeek(UM), Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale politile gegevensuitwisseling en digitale expertise.

2004-09 Martin Caminada (VU), For the Sake of the Argument; explorations into argument-based reasoning.

2004-10 Suzanne Kabel (UVA), Knowledge-rich indexing of learning-objects.

2004-11 Michel Klein (VU), Change Management for Distributed Ontologies.

2004-12 The Duy Bui (UT), Creating emotions and facial expressions for embodied agents.

2004-13 Wojciech Jamroga (UT), Using Multiple Models of Reality: On Agents who Know how to Play.

2004-14 Paul Harrenstein (UU), Logic in Conflict. Logical Explorations in Strategic Equilibrium.

2004-15 Arno Knobbe (UU), Multi-Relational Data Mining.

2004-16 Federico Divina (VU), Hybrid Genetic Relational Search for Inductive Learning.

2004-17 Mark Winands (UM), Informed Search in Complex Games.

2004-18 Vania Bessa Machado (UvA), Supporting the Construction of Qualitative Knowledge Models.

2004-19 Thijs Westerveld (UT), Using generative probabilistic models for multimedia retrieval.

2004-20 Madelon Evers (Nyenrode), Learning from Design: facilitating multidisciplinary design teams.

2005-01 Floor Verdenius (UVA), Methodological Aspects of Designing Induction-Based Applications.

2005-02 Erik van der Werf (UM)), AI techniques for the game of Go.

2005-03 Franc Grootjen (RUN), A Pragmatic Approach to the Conceptualisation of Language.

2005-04 Nirvana Meratnia (UT), Towards Database Support for Moving Object data.

2005-05 Gabriel Infante-Lopez (UVA), Two-Level Probabilistic Grammars for Natural Language Parsing.

2005-06 Pieter Spronck (UM), Adaptive Game AI.

2005-07 Flavius Frasincar (TUE), Hypermedia Presentation Generation for Semantic Web Information Systems.

2005-08 Richard Vdovjak (TUE), A Model-driven Approach for Building Distributed Ontology-based Web Applications.

2005-09 Jeen Broekstra (VU), Storage, Querying and Inferencing for Semantic Web Languages.

2005-10 Anders Bouwer (UVA), Explaining Behaviour: Using Qualitative Simulation in Interactive Learning Environments.

2005-11 Elth Ogston (VU), Agent Based Matchmaking and Clustering - A Decentralized Approach to Search.

2005-12 Csaba Boer (EUR), Distributed Simulation in Industry.

2005-13 Fred Hamburg (UL), Een Computermodel voor het Ondersteunen van Euthanasiebeslissingen.

2005-14 Borys Omelayenko (VU), Web-Service configuration on the Semantic Web; Exploring how semantics meets pragmatics.

2005-15 Tibor Bosse (VU), Analysis of the Dynamics of Cognitive Processes.

2005-16 Joris Graaumans (UU), Usability of XML Query Languages.

2005-17 Boris Shishkov (TUD), Software Specification Based on Re-usable Business Components.

2005-18 Danielle Sent (UU), Test-selection strategies for probabilistic networks.

2005-19 Michel van Dartel (UM), Situated Representation.

2005-20 Cristina Coteanu (UL), Cyber Consumer Law, State of the Art and Perspectives.

2005-21 Wijnand Derks (UT), Improving Concurrency and Recovery in Database Systems by Exploiting Application Semantics.

2006-01 Samuil Angelov (TUE), Foundations of B2B Electronic Contracting.

2006-02 Cristina Chisalita (VU), Contextual issues in the design and use of information technology in organizations.

2006-03 Noor Christoph (UVA), The role of metacognitive skills in learning to solve problems.

2006-04 Marta Sabou (VU), Building Web Service Ontologies.

2006-05 Cees Pierik (UU), Validation Techniques for Object-Oriented Proof Outlines.

2006-06 Ziv Baida (VU), Software-aided Service Bundling - Intelligent Methods & Tools for Graphical Service Modeling.

2006-07 Marko Smiljanic (UT), XML schema matching – balancing efficiency and effectiveness by means of clustering.

2006-08 Eelco Herder (UT), Forward, Back and Home Again - Analyzing User Behavior on the Web.

2006-09 Mohamed Wahdan (UM), Automatic Formulation of the Auditor's Opinion.

2006-10 Ronny Siebes (VU), Semantic Routing in Peer-to-Peer Systems.

2006-11 Joeri van Ruth (UT), Flattening Queries over Nested Data Types.

2006-12 Bert Bongers (VU), Interactivation - Towards an e-cology of people, our technological environment, and the arts.

2006-13 Henk-Jan Lebbink (UU), Dialogue and Decision Games for Information Exchanging Agents.

2006-14 Johan Hoorn (VU), Software Requirements: Update, Upgrade, Redesign - towards a Theory of Requirements Change.

2006-15 Rainer Malik (UU), CONAN: Text Mining in the Biomedical Domain.

2006-16 Carsten Riggelsen (UU), Approximation Methods for Efficient Learning of Bayesian Networks.

2006-17 Stacey Nagata (UU), User Assistance for Multitasking with Interruptions on a Mobile Device.

2006-18 Valentin Zhizhkun (UVA), Graph transformation for Natural Language Processing.

2006-19 Birna van Riemsdijk (UU), Cognitive Agent Programming: A Semantic Approach.

2006-20 Marina Velikova (UvT), Monotone models for prediction in data mining.

2006-21 Bas van Gils (RUN), Aptness on the Web.

2006-22 Paul de Vrieze (RUN), Fundaments of Adaptive Personalisation.

2006-23 Ion Juvina (UU), Development of Cognitive Model for Navigating on the Web.

2006-24 Laura Hollink (VU), Semantic Annotation for Retrieval of Visual Resources.

2006-25 Madalina Drugan (UU), Conditional log-likelihood MDL and Evolutionary MCMC.

2006-26 Vojkan Mihajlovic (UT), Score Region Algebra: A Flexible Framework for Structured Information Retrieval.

2006-27 Stefano Bocconi (CWI), Vox Populi: generating video documentaries from semantically annotated media repositories.

2006-28 Borkur Sigurbjornsson (UVA), Focused Information Access using XML Element Retrieval.

2007-01 Kees Leune (UvT), Access Control and Service-Oriented Architectures.

2007-02 Wouter Teepe (RUG), Reconciling Information Exchange and Confidentiality: A Formal Approach.

2007-03 Peter Mika (VU), Social Networks and the Semantic Web.

2007-04 Jurriaan van Diggelen (UU), Achieving Semantic Interoperability in Multi-agent Systems: a dialogue-based approach.

2007-05 Bart Schermer (UL), Software Agents, Surveillance, and the Right to Privacy: a Legislative Framework for Agent-enabled Surveillance.

2007-06 Gilad Mishne (UVA), Applied Text Analytics for Blogs.

2007-07 Natasa Jovanovic' (UT), To Whom It May Concern - Addressee Identification in Face-to-Face Meetings.

2007-08 Mark Hoogendoorn (VU), Modeling of Change in Multi-Agent Organizations.

2007-09 David Mobach (VU), Agent-Based Mediated Service Negotiation.

2007-10 Huib Aldewereld (UU), Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols.

2007-11 Natalia Stash (TUE), Incorporating Cognitive/Learning Styles in a General-Purpose Adaptive Hypermedia System.

2007-12 Marcel van Gerven (RUN), Bayesian Networks for Clinical Decision Support: A Rational Approach to Dynamic Decision-Making under Uncertainty.

2007-13 Rutger Rienks (UT), Meetings in Smart Environments; Implications of Progressing Technology.

2007-14 Niek Bergboer (UM), Context-Based Image Analysis.

2007-15 Joyca Lacroix (UM), NIM: a Situated Computational Memory Model.

2007-16 Davide Grossi (UU), Designing Invisible Handcuffs. Formal investigations in Institutions and Organizations for Multi-agent Systems.

2007-17 Theodore Charitos (UU), Reasoning with Dynamic Networks in Practice.

2007-18 Bart Orriens (UvT), On the development an management of adaptive business collaborations.

2007-19 David Levy (UM), Intimate relationships with artificial partners.

2007-20 Slinger Jansen (UU), Customer Configuration Updating in a Software Supply Network.

2007-21 Karianne Vermaas (UU), Fast diffusion and broadening use: A research on residential adoption and usage of broadband internet in the Netherlands between 2001 and 2005.

2007-22 Zlatko Zlatev (UT), Goal-oriented design of value and process models from patterns.

2007-23 Peter Barna (TUE), Specification of Application Logic in Web Information Systems.

2007-24 Georgina Ramrez Camps (CWI), Structural Features in XML Retrieval.

2007-25 Joost Schalken (VU), Empirical Investigations in Software Process Improvement.

2008-01 Katalin Boer-Sorbn (EUR), Agent-Based Simulation of Financial Markets: A modular, continuous-time approach.

2008-02 Alexei Sharpanskykh (VU), On Computer-Aided Methods for Modeling and Analysis of Organizations.

2008-03 Vera Hollink (UVA), Optimizing hierarchical menus: a usage-based approach.

2008-04 Ander de Keijzer (UT), Management of Uncertain Data - towards unattended integration.

2008-05 Bela Mutschler (UT), Modeling and simulating causal dependencies on process-aware information systems from a cost perspective.

2008-06 Arjen Hommersom (RUN), On the Application of Formal Methods to Clinical Guidelines, an Artificial Intelligence Perspective.

2008-07 Peter van Rosmalen (OU), Supporting the tutor in the design and support of adaptive e-learning.

2008-08 Janneke Bolt (UU), Bayesian Networks: Aspects of Approximate Inference.

2008-09 Christof van Nimwegen (UU), The paradox of the guided user: assistance can be counter-effective.

2008-10 Wauter Bosma (UT), Discourse oriented summarization.

2008-11 Vera Kartseva (VU), Designing Controls for Network Organizations: A Value-Based Approach.

2008-12 Jozsef Farkas (RUN), A Semiotically Oriented Cognitive Model of Knowledge Representation.

2008-13 Caterina Carraciolo (UVA), Topic Driven Access to Scientific Handbooks.

2008-14 Arthur van Bunningen (UT), Context-Aware Querying; Better Answers with Less Effort.

2008-15 Martijn van Otterlo (UT), The Logic of Adaptive Behavior: Knowledge Representation and Algorithms for the Markov Decision Process Framework in First-Order Domains.

2008-16 Henriette van Vugt (VU), Embodied agents from a user's perspective.

2008-17 Martin Op 't Land (TUD), Applying Architecture and Ontology to the Splitting and Allying of Enterprises.

2008-18 Guido de Croon (UM), Adaptive Active Vision.

2008-19 Henning Rode (UT), From Document to Entity Retrieval: Improving Precision and Performance of Focused Text Search.

2008-20 Rex Arendsen (UVA), Geen bericht, goed bericht. Een onderzoek naar de effecten van de introductie van elektronisch berichtenverkeer met de overheid op de administratieve lasten van bedrijven.

2008-21 Krisztian Balog (UVA), People Search in the Enterprise.

2008-22 Henk Koning (UU), Communication of IT-Architecture.

2008-23 Stefan Visscher (UU), Bayesian network models for the management of ventilator-associated pneumonia.

2008-24 Zharko Aleksovski (VU), Using background knowledge in ontology matching.

2008-25 Geert Jonker (UU), Efficient and Equitable Exchange in Air Traffic Management Plan Repair using Spender-signed Currency.

2008-26 Marijn Huijbregts (UT), Segmentation, Diarization and Speech Transcription: Surprise Data Unraveled.

2008-27 Hubert Vogten (OU), Design and Implementation Strategies for IMS Learning Design.

2008-28 Ildiko Flesch (RUN), On the Use of Independence Relations in Bayesian Networks.

2008-29 Dennis Reidsma (UT), Annotations and Subjective Machines - Of Annotators, Embodied Agents, Users, and Other Humans.

2008-30 Wouter van Atteveldt (VU), Semantic Network Analysis: Techniques for Extracting, Representing and Querying Media Content.

2008-31 Loes Braun (UM), Pro-Active Medical Information Retrieval.

2008-32 Trung H. Bui (UT), Toward Affective Dialogue Management using Partially Observable Markov Decision Processes.

2008-33 Frank Terpstra (UVA), Scientific Workflow Design; theoretical and practical issues.

2008-34 Jeroen de Knijf (UU), Studies in Frequent Tree Mining.

2008-35 Ben Torben Nielsen (UvT), Dendritic morphologies: function shapes structure.

2009-01 Rasa Jurgelenaite (RUN), Symmetric Causal Independence Models.

2009-02 Willem Robert van Hage (VU), Evaluating Ontology-Alignment Techniques.

2009-03 Hans Stol (UvT), A Framework for Evidence-based Policy Making Using IT.

2009-04 Josephine Nabukenya (RUN), Improving the Quality of Organisational Policy Making using Collaboration Engineering.

2009-05 Sietse Overbeek (RUN), Bridging Supply and Demand for Knowledge Intensive Tasks - Based on Knowledge, Cognition, and Quality.

2009-06 Muhammad Subianto (UU), Understanding Classification.

2009-07 Ronald Poppe (UT), Discriminative Vision-Based Recovery and Recognition of Human Motion.

2009-08 Volker Nannen (VU), Evolutionary Agent-Based Policy Analysis in Dynamic Environments.

2009-09 Benjamin Kanagwa (RUN), Design, Discovery and Construction of Service-oriented Systems.

2009-10 Jan Wielemaker (UVA), Logic programming for knowledge-intensive interactive applications.

2009-11 Alexander Boer (UVA), Legal Theory, Sources of Law & the Semantic Web.

2009-12 Peter Massuthe (TUE, Humboldt-Universitaet zu Berlin), Operating Guidelines for Services.

2009-13 Steven de Jong (UM), Fairness in Multi-Agent Systems.

2009-14 Maksym Korotkiy (VU), From ontology-enabled services to service-enabled ontologies (making ontologies work in e-science with ONTO-SOA).

2009-15 Rinke Hoekstra (UVA), Ontology Representation - Design Patterns and Ontologies that Make Sense.

2009-16 Fritz Reul (UvT), New Architectures in Computer Chess.

2009-17 Laurens van der Maaten (UvT), Feature Extraction from Visual Data.

2009-18 Fabian Groffen (CWI), Armada, An Evolving Database System.

2009-19 Valentin Robu (CWI), Modeling Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets.

2009-20 Bob van der Vecht (UU), Adjustable Autonomy: Controling Influences on Decision Making.

2009-21 Stijn Vanderlooy (UM), Ranking and Reliable Classification.

2009-22 Pavel Serdyukov (UT), Search For Expertise: Going beyond direct evidence.

2009-23 Peter Hofgesang (VU), Modelling Web Usage in a Changing Environment.

2009-24 Annerieke Heuvelink (VUA), Cognitive Models for Training Simulations.

2009-25 Alex van Ballegooij (CWI), RAM: Array Database Management through Relational Mapping.

2009-26 Fernando Koch (UU), An Agent-Based Model for the Development of Intelligent Mobile Services.

2009-27 Christian Glahn (OU), Contextual Support of social Engagement and Reflection on the Web.

2009-28 Sander Evers (UT), Sensor Data Management with Probabilistic Models.

2009-29 Stanislav Pokraev (UT), Model-Driven Semantic Integration of Service-Oriented Applications.

2009-30 Marcin Zukowski (CWI), Balancing vectorized query execution with bandwidth-optimized storage.

2009-31 Sofiya Katrenko (UVA), A Closer Look at Learning Relations from Text.

2009-32 Rik Farenhorst (VU), and Remco de Boer (VU).

Architectural Knowledge Management: Supporting Architects and Auditors **2009-33** Khiet Truong (UT), How Does Real Affect Affect Affect Recognition In Speech?.

2009-34 Inge van de Weerd (UU), Advancing in Software Product Management: An Incremental Method Engineering Approach.

2009-35 Wouter Koelewijn (UL), Privacy en Politiegegevens; Over geautomatiseerde normatieve informatie-uitwisseling.

2009-36 Marco Kalz (OUN), Placement Support for Learners in Learning Networks.

2009-37 Hendrik Drachsler (OUN), Navigation Support for Learners in Informal Learning Networks.

2009-38 Riina Vuorikari (OU), Tags and self-organisation: a metadata ecology for learning resources in a multilingual context.

2009-39 Christian Stahl (TUE, Humboldt-Universitaet zu Berlin), Service Substitution – A Behavioral Approach Based on Petri Nets.

2009-40 Stephan Raaijmakers (UvT), Multinomial Language Learning: Investigations into the Geometry of Language.

2009-41 Igor Berezhnyy (UvT), Digital Analysis of Paintings.

2009-42 Toine Bogers (UvT), Recommender Systems for Social Bookmarking.

2009-43 Virginia Nunes Leal Franqueira (UT), Finding Multi-step Attacks in Computer Networks using Heuristic Search and Mobile Ambients.

2009-44 Roberto Santana Tapia (UT), Assessing Business-IT Alignment in Networked Organizations.

2009-45 Jilles Vreeken (UU), Making Pattern Mining Useful.

2009-46 Loredana Afanasiev (UvA), Querying XML: Benchmarks and Recursion.

2010-01 Matthijs van Leeuwen (UU), Patterns that Matter.

2010-02 Ingo Wassink (UT), Work flows in Life Science.

2010-03 Joost Geurts (CWI), A Document Engineering Model and Processing Framework for Multimedia documents.

2010-04 Olga Kulyk (UT), Do You Know What I Know? Situational Awareness of Co-located Teams in Multidisplay Environments.

2010-05 Claudia Hauff (UT), Predicting the Effectiveness of Queries and Retrieval Systems.

2010-06 Sander Bakkes (UvT), Rapid Adaptation of Video Game AI.

2010-07 Wim Fikkert (UT), Gesture interaction at a Distance.

2010-08 Krzysztof Siewicz (UL), Towards an Improved Regulatory Framework of Free Software. Protecting user freedoms in a world of software communities and eGovernments.

2010-09 Hugo Kielman (UL), A Politiele gegevensverwerking en Privacy, Naar een effectieve waarborging.

2010-10 Rebecca Ong (UL), Mobile Communication

and Protection of Children.

2010-11 Adriaan Ter Mors (TUD), The world according to MARP: Multi-Agent Route Planning.

2010-12 Susan van den Braak (UU), Sensemaking software for crime analysis.

2010-13 Gianluigi Folino (RUN), High Performance Data Mining using Bio-inspired techniques.

2010-14 Sander van Splunter (VU), Automated Web Service Reconfiguration.

2010-15 Lianne Bodenstaff (UT), Managing Dependency Relations in Inter-Organizational Models.

2010-16 Sicco Verwer (TUD), Efficient Identification of Timed Automata, theory and practice.

2010-17 Spyros Kotoulas (VU), Scalable Discovery of Networked Resources: Algorithms, Infrastructure, Applications.

2010-18 Charlotte Gerritsen (VU), Caught in the Act: Investigating Crime by Agent-Based Simulation.

2010-19 Henriette Cramer (UvA), People's Responses to Autonomous and Adaptive Systems.

2010-20 Ivo Swartjes (UT), Whose Story Is It Anyway? How Improv Informs Agency and Authorship of Emergent Narrative.

2010-21 Harold van Heerde (UT), Privacy-aware data management by means of data degradation.

2010-22 Michiel Hildebrand (CWI), End-user Support for Access to Heterogeneous Linked Data.

2010-23 Bas Steunebrink (UU), The Logical Structure of Emotions.

2010-24 Dmytro Tykhonov (VU), Designing Generic and Efficient Negotiation Strategies.

2010-25 Zulfiqar Ali Memon (VU), Modelling Human-Awareness for Ambient Agents: A Human Mindreading Perspective.

2010-26 Ying Zhang (CWI), XRPC: Efficient Distributed Query Processing on Heterogeneous XQuery Engines.

2010-27 Marten Voulon (UL), Automatisch contracteren.

2010-28 Arne Koopman (UU), Characteristic Relational Patterns.

2010-29 Stratos Idreos(CWI), Database Cracking: Towards Auto-tuning Database Kernels.

2010-30 Marieke van Erp (UvT), Accessing Natural History - Discoveries in data cleaning, structuring, and retrieval.

2010-31 Victor de Boer (UVA), Ontology Enrichment from Heterogeneous Sources on the Web.

2010-32 Marcel Hiel (UvT), An Adaptive Service Oriented Architecture: Automatically solving Interoperability Problems.

2010-33 Robin Aly (UT), Modeling Representation Uncertainty in Concept-Based Multimedia Retrieval.

2010-34 Teduh Dirgahayu (UT), Interaction Design in Service Compositions.

2010-35 Dolf Trieschnigg (UT), Proof of Concept: Concept-based Biomedical Information Retrieval.

2010-36 Jose Janssen (OU), Paving the Way for Lifelong Learning; Facilitating competence development through a learning path specification.

2010-37 Niels Lohmann (TUE), Correctness of services and their composition.

2010-38 Dirk Fahland (TUE), From Scenarios to components.

2010-39 Ghazanfar Farooq Siddiqui (VU), Integrative modeling of emotions in virtual agents.

2010-40 Mark van Assem (VU), Converting and Integrating Vocabularies for the Semantic Web.

2010-41 Guillaume Chaslot (UM), Monte-Carlo Tree Search.

2010-42 Sybren de Kinderen (VU), Needs-driven service bundling in a multi-supplier setting - the computational e3-service approach.

2010-43 Peter van Kranenburg (UU), A Computational Approach to Content-Based Retrieval of Folk Song Melodies.

2010-44 Pieter Bellekens (TUE), An Approach towards Context-sensitive and User-adapted Access to Heterogeneous Data Sources, Illustrated in the Television Domain.

2010-45 Vasilios Andrikopoulos (UvT), A theory and model for the evolution of software services.

2010-46 Vincent Pijpers (VU), e3alignment: Exploring Inter-Organizational Business-ICT Alignment.

2010-47 Chen Li (UT), Mining Process Model Variants: Challenges, Techniques, Examples.

2010-48 , Withdrawn.

2010-49 Jahn-Takeshi Saito (UM), Solving difficult game positions.

2010-50 Bouke Huurnink (UVA), Search in Audiovisual Broadcast Archives.

2010-51 Alia Khairia Amin (CWI), Understanding and supporting information seeking tasks in multiple sources.

2010-52 Peter-Paul van Maanen (VU), Adaptive Support for Human-Computer Teams: Exploring the Use of Cognitive Models of Trust and Attention.

2010-53 Edgar Meij (UVA), Combining Concepts and Language Models for Information Access.

2011-01 Botond Cseke (RUN), Variational Algorithms for Bayesian Inference in Latent Gaussian Models.

2011-02 Nick Tinnemeier(UU), Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language.

2011-03 Jan Martijn van der Werf (TUE), Compositional Design and Verification of Component-Based Information Systems.

2011-04 Hado van Hasselt (UU), Insights in Reinforcement Learning; Formal analysis and empirical evaluation of temporal-difference learning algorithms.

2011-05 Base van der Raadt (VU), Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline.

2011-06 Yiwen Wang (TUE), Semantically-Enhanced Recommendations in Cultural Heritage.

2011-07 Yujia Cao (UT), Multimodal Information Presentation for High Load Human Computer Interaction.

2011-08 Nieske Vergunst (UU), BDI-based Generation of Robust Task-Oriented Dialogues.

2011-09 Tim de Jong (OU), Contextualised Mobile Media for Learning.

2011-10 Bart Bogaert (UvT), Cloud Content Contention.

2011-11 Dhaval Vyas (UT), Designing for Awareness: An Experience-focused HCI Perspective.

2011-12 Carmen Bratosin (TUE), Grid Architecture for Distributed Process Mining.

2011-13 Xiaoyu Mao (UvT), Airport under Control. Multiagent Scheduling for Airport Ground Handling.

2011-14 Milan Lovric (EUR), Behavioral Finance and Agent-Based Artificial Markets.

2011-15 Marijn Koolen (UvA), The Meaning of Structure: the Value of Link Evidence for Information Retrieval.

2011-16 Maarten Schadd (UM), Selective Search in Games of Different Complexity.

2011-17 Jiyin He (UVA), Exploring Topic Structure: Coherence, Diversity and Relatedness.

2011-18 Mark Ponsen (UM), Strategic Decision-Making in complex games.

2011-19 Ellen Rusman (OU), The Mind ' s Eye on Personal Profiles.

2011-20 Qing Gu (VU), Guiding service-oriented software engineering - A view-based approach.

2011-21 Linda Terlouw (TUD), Modularization and Specification of Service-Oriented Systems.

2011-22 Junte Zhang (UVA), System Evaluation of Archival Description and Access.

2011-23 Wouter Weerkamp (UVA), Finding People and their Utterances in Social Media.

2011-24 Herwin van Welbergen (UT), Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior.

2011-25 Syed Waqar ul Qounain Jaffry (VU)), Analysis and Validation of Models for Trust Dynamics.

2011-26 Matthijs Aart Pontier (VU), Virtual Agents for Human Communication - Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots.

2011-27 Aniel Bhulai (VU), Dynamic website optimization through autonomous management of design patterns.

2011-28 Rianne Kaptein(UVA), Effective Focused Retrieval by Exploiting Query Context and Document Structure.

2011-29 Faisal Kamiran (TUE), Discrimination-aware Classification.

2011-30 Egon van den Broek (UT), Affective Signal Processing (ASP): Unraveling the mystery of emotions.

2011-31 Ludo Waltman (EUR), Computational and Game-Theoretic Approaches for Modeling Bounded Rationality.

2011-32 Nees-Jan van Eck (EUR), Methodological Advances in Bibliometric Mapping of Science.

2011-33 Tom van der Weide (UU), Arguing to Motivate Decisions.

2011-34 Paolo Turrini (UU), Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations.

2011-35 Maaike Harbers (UU), Explaining Agent Behavior in Virtual Training.

2011-36 Erik van der Spek (UU), Experiments in serious game design: a cognitive approach.

2011-37 Adriana Burlutiu (RUN), Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference.

2011-38 Nyree Lemmens (UM), Bee-inspired Distributed Optimization.

2011-39 Joost Westra (UU), Organizing Adaptation using Agents in Serious Games.

2011-40 Viktor Clerc (VU), Architectural Knowledge Management in Global Software Development.

2011-41 Luan Ibraimi (UT), Cryptographically Enforced Distributed Data Access Control.

2011-42 Michal Sindlar (UU), Explaining Behavior through Mental State Attribution.

2011-43 Henk van der Schuur (UU), Process Improvement through Software Operation Knowledge.

2011-44 Boris Reuderink (UT), Robust Brain-Computer Interfaces.

2011-45 Herman Stehouwer (UvT), Statistical Language Models for Alternative Sequence Selection.

2011-46 Beibei Hu (TUD), Towards Contextualized Information Delivery: A Rule-based Architecture for the Domain of Mobile Police Work.

2011-47 Azizi Bin Ab Aziz(VU), Exploring Computational Models for Intelligent Support of Persons with Depression.

2011-48 Mark Ter Maat (UT), Response Selection and Turn-taking for a Sensitive Artificial Listening Agent.

2011-49 Andreea Niculescu (UT), Conversational interfaces for task-oriented spoken dialogues: design aspects influencing interaction quality.

2012-01 Terry Kakeeto (UvT), Relationship Marketing for SMEs in Uganda.

2012-02 Muhammad Umair(VU), Adaptivity, emotion, and Rationality in Human and Ambient Agent Models.

2012-03 Adam Vanya (VU), Supporting Architecture Evolution by Mining Software Repositories.

2012-04 Jurriaan Souer (UU), Development of Content Management System-based Web Applications.

2012-05 Marijn Plomp (UU), Maturing Interorganisational Information Systems.

2012-06 Wolfgang Reinhardt (OU), Awareness Support for Knowledge Workers in Research Networks.

2012-07 Rianne van Lambalgen (VU), When the Going Gets Tough: Exploring Agent-based Models of Human Performance under Demanding Conditions.

2012-08 Gerben de Vries (UVA), Kernel Methods for Vessel Trajectories.

2012-09 Ricardo Neisse (UT), Trust and Privacy Management Support for Context-Aware Service Platforms.

2012-10 David Smits (TUE), Towards a Generic Distributed Adaptive Hypermedia Environment.

2012-11 J.C.B. Rantham Prabhakara (TUE), Process Mining in the Large: Preprocessing, Discovery, and Diagnostics.

2012-12 Kees van der Sluijs (TUE), Model Driven Design and Data Integration in Semantic Web Information Systems.

2012-13 Suleman Shahid (UvT), Fun and Face: Exploring non-verbal expressions of emotion during playful interactions.

2012-14 Evgeny Knutov(TUE), Generic Adaptation Framework for Unifying Adaptive Web-based Systems.

2012-15 Natalie van der Wal (VU), Social Agents. Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes.

2012-16 Fiemke Both (VU), Helping people by understanding them - Ambient Agents supporting task execution and depression treatment.

2012-17 Amal Elgammal (UvT), Towards a Comprehensive Framework for Business Process Compliance.

2012-18 Eltjo Poort (VU), Improving Solution Architecting Practices.

2012-19 Helen Schonenberg (TUE), What's Next? Operational Support for Business Process Execution.

2012-20 Ali Bahramisharif (RUN), Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer Interfacing.

2012-21 Roberto Cornacchia (TUD), Querying Sparse Matrices for Information Retrieval.

2012-22 Thijs Vis (UvT), Intelligence, politie en veiligheidsdienst: verenigbare grootheden?.

2012-23 Christian Muehl (UT), Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology of Affect during Human Media Interaction.

2012-24 Laurens van der Werff (UT), Evaluation of Noisy Transcripts for Spoken Document Retrieval.

2012-25 Silja Eckartz (UT), Managing the Business Case Development in Inter-Organizational IT Projects: A Methodology and its Application.

2012-26 Emile de Maat (UVA), Making Sense of Legal Text.

2012-27 Hayrettin Gurkok (UT), Mind the Sheep! User Experience Evaluation & Brain-Computer Interface Games.

2012-28 Nancy Pascall (UvT), Engendering Technology Empowering Women.

2012-29 Almer Tigelaar (UT), Peer-to-Peer Information Retrieval.

2012-30 Alina Pommeranz (TUD), Designing Human-Centered Systems for Reflective Decision Making.

2012-31 Emily Bagarukayo (RUN), A Learning by Construction Approach for Higher Order Cognitive Skills Improvement, Building Capacity and Infrastructure.

2012-32 Wietske Visser (TUD), Qualitative multi-criteria preference representation and reasoning.

2012-33 Rory Sie (OUN), Coalitions in Cooperation Networks (COCOON).

2012-34 Pavol Jancura (RUN), Evolutionary analysis in PPI networks and applications.

2012-35 Evert Haasdijk (VU), Never Too Old To Learn – On-line Evolution of Controllers in Swarm- and Modular Robotics.

2012-36 Denis Ssebugwawo (RUN), Analysis and Evaluation of Collaborative Modeling Processes.

2012-37 Agnes Nakakawa (RUN), A Collaboration Process for Enterprise Architecture Creation.

2012-38 Selmar Smit (VU), Parameter Tuning and Scientific Testing in Evolutionary Algorithms.

2012-39 Hassan Fatemi (UT), Risk-aware design of value and coordination networks.

2012-40 Agus Gunawan (UvT), Information Access for SMEs in Indonesia.

2012-41 Sebastian Kelle (OU), Game Design Patterns for Learning.

2012-42 Dominique Verpoorten (OU), Reflection Amplifiers in self-regulated Learning.

2012-43 , Withdrawn.

2012-44 Anna Tordai (VU), On Combining Alignment Techniques.

2012-45 Benedikt Kratz (UvT), A Model and Language for Business-aware Transactions.

2012-46 Simon Carter (UVA), Exploration and Exploitation of Multilingual Data for Statistical Machine Translation.

2012-47 Manos Tsagkias (UVA), Mining Social Media: Tracking Content and Predicting Behavior.

2012-48 Jorn Bakker (TUE), Handling Abrupt Changes in Evolving Time-series Data.

2012-49 Michael Kaisers (UM), Learning against Learning - Evolutionary dynamics of reinforcement learning algorithms in strategic interactions.

2012-50 Steven van Kervel (TUD), Ontologogy driven Enterprise Information Systems Engineering.

2012-51 Jeroen de Jong (TUD), Heuristics in Dynamic Sceduling; a practical framework with a case study in elevator dispatching.

2013-01 Viorel Milea (EUR), News Analytics for Financial Decision Support.

2013-02 Erietta Liarou (CWI), MonetDB/DataCell: Leveraging the Column-store Database Technology for Efficient and Scalable Stream Processing.

2013-03 Szymon Klarman (VU), Reasoning with Contexts in Description Logics.

2013-04 Chetan Yadati(TUD), Coordinating autonomous planning and scheduling.

2013-05 Dulce Pumareja (UT), Groupware Requirements Evolutions Patterns.

2013-06 Romulo Goncalves(CWI), The Data Cyclotron: Juggling Data and Queries for a Data Warehouse Audience.

2013-07 Giel van Lankveld (UvT), Quantifying Individual Player Differences.

2013-08 Robbert-Jan Merk(VU), Making enemies: cognitive modeling for opponent agents in fighter pilot simulators.

2013-09 Fabio Gori (RUN), Metagenomic Data Analysis: Computational Methods and Applications.

2013-10 Jeewanie Jayasinghe Arachchige(UvT), A Unified Modeling Framework for Service Design.

2013-11 Evangelos Pournaras(TUD), Multi-level Reconfigurable Self-organization in Overlay Services.

2013-12 Marian Razavian(VU), Knowledge-driven Migration to Services.

2013-13 Mohammad Safiri(UT), Service Tailoring: User-centric creation of integrated IT-based homecare services to support independent living of elderly.

2013-14 Jafar Tanha (UVA), Ensemble Approaches to Semi-Supervised Learning Learning.

2013-15 Daniel Hennes (UM), Multiagent Learning - Dynamic Games and Applications.

2013-16 Eric Kok (UU), Exploring the practical benefits of argumentation in multi-agent deliberation.

2013-17 Koen Kok (VU), The PowerMatcher: Smart Coordination for the Smart Electricity Grid.

2013-18 Jeroen Janssens (UvT), Outlier Selection and One-Class Classification.

2013-19 Renze Steenhuizen (TUD), Coordinated Multi-Agent Planning and Scheduling.

2013-20 Katja Hofmann (UvA), Fast and Reliable Online Learning to Rank for Information Retrieval.

2013-21 Sander Wubben (UvT), Text-to-text generation by monolingual machine translation.

2013-22 Tom Claassen (RUN), Causal Discovery and Logic.

2013-23 Patricio de Alencar Silva(UvT), Value Activity Monitoring.

2013-24 Haitham Bou Ammar (UM), Automated Transfer in Reinforcement Learning.

2013-25 Agnieszka Anna Latoszek-Berendsen (UM), Intention-based Decision Support. A new way of representing and implementing clinical guidelines in a Decision Support System.

2013-26 Alireza Zarghami (UT), Architectural Support for Dynamic Homecare Service Provisioning.

2013-27 Mohammad Huq (UT), Inference-based Framework Managing Data Provenance.

2013-28 Frans van der Sluis (UT), When Complexity becomes Interesting: An Inquiry into the Information eXperience.

2013-29 Iwan de Kok (UT), Listening Heads.

2013-30 Joyce Nakatumba (TUE), Resource-Aware Business Process Management: Analysis and Support.

2013-31 Dinh Khoa Nguyen (UvT), Blueprint Model and Language for Engineering Cloud Applications.

2013-32 Kamakshi Rajagopal (OUN), Networking For Learning; The role of Networking in a Lifelong Learner's Professional Development.

2013-33 Qi Gao (TUD), User Modeling and Personalization in the Microblogging Sphere.

2013-34 Kien Tjin-Kam-Jet (UT), Distributed Deep Web Search.

2013-35 Abdallah El Ali (CWI/UvA), Minimal Mobile Human Computer Interaction.

2013-36 Than Lam Hoang (TUe), Pattern Mining in

Data Streams.

2013-37 Dirk Brner (OUN), Ambient Learning Displays.

2013-38 Eelco den Heijer (VU), Autonomous Evolutionary Art.

2013-39 Joop de Jong (TUD), A Method for Enterprise Ontology based Design of Enterprise Information Systems.

2013-40 Pim Nijssen (UM), Monte-Carlo Tree Search for Multi-Player Games.

2013-41 Jochem Liem (UVA), Supporting the Conceptual Modelling of Dynamic Systems: A Knowledge Engineering Perspective on Qualitative Reasoning.

2013-42 Lon Planken (TUD), Algorithms for Simple Temporal Reasoning.

2013-43 Marc Bron (UVA), Exploration and Contextualization through Interaction and Concepts.

2014-01 Nicola Barile (UU), Studies in Learning Monotone Models from Data.

2014-02 Fiona Tuliyano (RUN), Combining System Dynamics with a Domain Modeling Method.

2014-03 Sergio Raul Duarte Torres (UT), Information Retrieval for Children: Search Behavior and Solutions.

2014-04 Hanna Jochmann-Mannak (UT), Websites for children: search strategies and interface design - Three studies on children's search performance and evaluation.

2014-05 Jurriaan van Reijsen (UU), Knowledge Perspectives on Advancing Dynamic Capability.

2014-06 Damian Tamburri (VU), Supporting Networked Software Development.

2014-07 Arya Adriansyah (TUE), Aligning Observed and Modeled Behavior.

2014-08 Samur Araujo (TUD), Data Integration over Distributed and Heterogeneous Data Endpoints.

2014-09 Philip Jackson (UvT), Toward Human-Level Artificial Intelligence: Representation and Computation of Meaning in Natural Language.

2014-10 Ivan Salvador Razo Zapata (VU), Service Value Networks.

2014-11 Janneke van der Zwaan (TUD), An Empathic Virtual Buddy for Social Support.

2014-12 Willem van Willigen (VU), Look Ma, No Hands: Aspects of Autonomous Vehicle Control.

2014-13 Arlette van Wissen (VU), Agent-Based Support for Behavior Change: Models and Applications in Health and Safety Domains.

2014-14 Yangyang Shi (TUD), Language Models With Meta-information.

2014-15 Natalya Mogles (VU), Agent-Based Analysis and Support of Human Functioning in Complex Socio-Technical Systems: Applications in Safety and Healthcare.

2014-16 Krystyna Milian (VU), Supporting trial recruitment and design by automatically interpreting eligibility criteria.

2014-17 Kathrin Dentler (VU), Computing healthcare quality indicators automatically: Secondary Use of Patient Data and Semantic Interoperability.

2014-18 Mattijs Ghijsen (VU), Methods and Models for the Design and Study of Dynamic Agent Organizations.

2014-19 Vincius Ramos (TUE), Adaptive Hypermedia Courses: Qualitative and Quantitative Evaluation and Tool Support.

2014-20 Mena Habib (UT), Named Entity Extraction and Disambiguation for Informal Text: The Missing Link.

2014-21 Kassidy Clark (TUD), Negotiation and Monitoring in Open Environments.

2014-22 Marieke Peeters (UT), Personalized Educational Games - Developing agent-supported scenario-based training.

2014-23 Eleftherios Sidirourgos (UvA/CWI), Space Efficient Indexes for the Big Data Era.

2014-24 Davide Ceolin (VU), Trusting Semi-structured Web Data.

2014-25 Martijn Lappenschaar (RUN), New network models for the analysis of disease interaction.

2014-26 Tim Baarslag (TUD), What to Bid and When to Stop.

2014-27 Rui Jorge Almeida (EUR), Conditional Density Models Integrating Fuzzy and Probabilistic Representations of Uncertainty.

2014-28 Anna Chmielowiec (VU), Decentralized k-Clique Matching.

2014-29 Jaap Kabbedijk (UU), Variability in Multi-Tenant Enterprise Software.

2014-30 Peter de Kock Berenschot (UvT), Anticipating Criminal Behaviour.

2014-31 Leo van Moergestel (UU), Agent Technology in Agile Multiparallel Manufacturing and Product Support.

2014-32 Naser Ayat (UVA), On Entity Resolution in Probabilistic Data.