# An Integrated Approach to Improve Data Quality

# AN INTEGRATED APPROACH TO IMPROVE DATA QUALITY

BY

SAMIR AL-JANABI, M.Eng.

a thesis

Submitted to the Department of Computing and Software

and the School of Graduate Studies

of McMaster University

in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

| | |
|---|---|
| Doctor of Philosophy (2016) | McMaster University |
| (Computing and Software) | Hamilton, Ontario, Canada |

| | |
|---|---|
| TITLE: | An Integrated Approach to Improve Data Quality |
| AUTHOR: | Samir Al-janabi |
| | M.Eng., (Software Engineering), |
| | McMaster University, Hamilton, Ontario, Canada |
| SUPERVISOR: | Dr. Ryszard Janicki |
| NUMBER OF PAGES: | xiv, 129 |

*To every person who taught me something useful*

# Abstract

A huge quantity of data is created and saved everyday in databases from different types of data sources, including financial data, web log data, sensor data, and human input. Information technology enables organizations to collect and store large amounts of data in databases. Different organizations worldwide use data to support their activities through various applications. Issues in data quality such as duplicate records, inaccurate data, violations of integrity constraints, and outdated data are common in databases. Thus, data in databases are often unclean. Such issues in data quality might cost billions of dollars annually and might have severe consequences on critical tasks such as analysis, decision making, and planning. Data cleaning processes are required to detect and correct errors in the unclean data.

Despite the fact that there are multiple quality issues, current data cleaning techniques generally deal with only one or two aspects of quality. The techniques assume either the availability of master data, or training data, or the involvement of users in data cleaning. For instance, users might manually place confidence scores that represent the correctness of the values of data or they may be consulted about the repairs. In addition, the techniques may depend on high-quality master data or pre-labeled training data to fix errors. However, relying on human effort to correct errors is expensive, and master data or training data are not always available. These factors make it challenging to discover which values have issues, thereby making it difficult to fix the data (e.g., merging several duplicate records into a single representative record).

To address these problems in data cleaning, we propose algorithms that integrate multiple data quality issues in the cleaning. In this thesis, we apply this approach in the context of multiple data quality issues where errors in data are introduced from multiple causes. The issues include duplicate records, violations of integrity constraints, inaccurate data, and outdated data. We fix these issues holistically, without a need for human manual interaction, master data, or training data.

We propose an algorithm to tackle the problem of data cleaning. We concentrate on issues in data quality including duplicate records, violations of integrity constraints, and inaccurate data. We utilize the embedded density information in data to eliminate duplicates based on data density, where tuples that are close to each other are packed together. Density information enables us to reduce manual user interaction in the deduplication process, and the dependency on master data or training data. To resolve inconsistency in duplicate records, we present a weight model to automatically assign confidence scores that are based on the density of data. We consider the inconsistent data in terms of violations with respect to a set of functional dependencies (FDs). We present a cost model for data repair that is based on the weight model. To resolve inaccurate data in duplicate records, we measure the relatedness of the words of the attributes in the duplicate records based on hierarchical clustering.

In the context of integrating the fix of outdated data and inaccurate data in duplicate elimination, we propose an algorithm for data cleaning by introducing techniques based on corroboration, i.e. taking into consideration the trustworthiness of the attribute values. The algorithm integrates data deduplication with data currency and accuracy. We utilize the density information embedded inside the tuples in order to guide the cleaning process to fix multiple data quality issues. By using density information in corroboration, we reduce relying on manual user interaction, and the dependency on master data or training data.

# Acknowledgements

First and foremost, I wish to express my deepest gratitude to my supervisor Dr. Ryszard Janicki, for the opportunity of working with him, and for his support and guidance throughout the development and advancement of my research. He has been always available to help and offer wise advice. He has taught me how to be a thorough and better researcher.

I am very thankful to the members of my supervisory committee: Dr. Emil Sekerinski and Dr. Franya Franek for their helpful suggestions, instructions, discussions, and insightful comments on my research.

I would like also to thank other faculty members of McMaster University, most notably Dr. Ivan Bruha for his help and support.

Last, but not least, I would like to thank my family members for their ongoing support. They have given so much without expecting anything in return.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

Information technology enables organizations to collect and store large amounts of data, which are processed and utilized to aid decision making. Data are used in various organizational activities and in all levels of management, from top to bottom. The importance of data quality should be understood by all, in order to motivate persons to actively participate in the assurance of data quality. In terms of databases, *data quality* concerns the caliber of data values and instances, and the dimensions that can be used to measure this caliber (Batini and Scannapieca, 2006). Some of the most common basic tenets are that data of high quality can increase the satisfaction of customers and can improve profits (Lee *et al.*, 2006). High quality data can have a positive impact on decisions, while low quality data can have the reverse effect. Examples of negative impacts include increases in penalties and costs of operations, increases in workloads, and processing time; improper decision-making; and risk and compliance impacts associated with fraud, credit assessment, investment risks, and compliance with government regulations (Loshin, 2010). The risks of bad data increase proportional to data volume (Watts *et al.*, 2009). Problems in data quality can cause increased costs and lower customer satisfaction. Therefore, cleaning tools are important to any business' data models.

There are several main data quality aspects in which issues would degrade quality. Data quality aspects may be characterized in a number of ways, such as identifiability, accuracy, currency, consistency, and completeness. *Identifiability* refers to the unique naming and representation of core conceptual objects and also the capability to link instances of data that contain entity data together based on the identification of attribute values (Loshin, 2010). For example, in a table, identifiability refers to the ability to uniquely locate a real-world entity using a set of attribute values. Data in databases may become impaired and unclean because of different reasons such as outdated values, duplicate records, missing values, inaccurate values, or violations of integrity constraints (Fan and Geerts, 2012). Given an identified source of correct information, the degree to which the values in data agree with this source is considered *data accuracy* (Loshin, 2010). *Data currency* is defined by (Redman, 1996) as the degree to which a datum is up-to-date. *Data consistency* refers to the integrity and validity of data that represent real-world entities, and *Information completeness* characterizes the extent to which a database has complete information in order to answer queries (Fan and Geerts, 2012). The *data cleaning process* refers to the detection and correction of erroneous values in data, with the aim of maintaining high quality data.

In the following, we summarize the main challenges that arise when implementing our density-based data cleaning approach, in addition to the drawbacks in the previous data cleaning techniques.

- **Fixing multiple data quality issues.** Previous techniques in data cleaning mainly consider fixing one or two issues. For example, repairing inconsistent data quality issue w.r.t. a set of integrity constraints, or fixing unclean data that has duplicate records issue caused by outdated data. In reality, data have multiple quality issues that need to be tackled holistically.

- **Handling a variety of causes of errors.** As there is an assortment of data quality issues, consequently, there is also a variety of causes for these issues. Multiple data sources that are integrated may cause duplicate records in the case where data have been entered using different standards. Transcription errors may cause inconsistent or inaccurate data records. Over time, data may become stale or inaccurate. Generally, existing data cleaning techniques handle one or two of the causes. For example, duplicate records that have outdated data caused over time, or inconsistent data due to transcription errors. In reality, dealing with multiple data quality issues requires the consideration of multiple causes of errors.

- **Manual user interaction-independent data cleaning.** Some previous techniques in duplicate elimination such as active learning, assume that users can manually interact with the data cleaning system to provide help in order to guide the cleaning process. In reality, databases are usually large in size, and relying on users to manually guide the process or to give suggestions is expensive, especially in terms of the cost of time.

- **Master data-independent data cleaning.** Some previous techniques in data cleaning assume the availability of a master data that contains complete information about the core entities of an organization (e.g., employees, customers, and products). In reality, such complete master data may not be available.

- **Training data-independent data cleaning.** Some previous techniques in duplicate elimination such as supervised learning, assume the availability of a training dataset that has pre-labeled duplicates, where the training data should have the variety and distribution of errors that are common to data in practice. In reality, it is hard to obtain such accurate quality of pre-labeled data.

In this thesis, our objective is to fix multiple data quality issues where errors are introduced due to multiple causes, without relying on manual user interaction, and without assuming the availability of master data or training data. We believe that density of data can provide information that may guide the cleaning process. We first investigated solutions to multiple data quality issues including duplicate records, inconsistent data, inaccurate data, and outdated data that are cause by multiple causes of errors. Second, we developed models and algorithms that address these issues without relying on manual user interaction, master data, or training data.

## 1.1 Duplicate Records Elimination

Duplicate records happen when tuples in one or more relation refer to the same real-world entity. Duplicate records do not need to be identical but it is expected that they are highly similar. Errors are introduced in duplicate records as the result of transcription errors, lack of standard formats, incomplete data, or any combination of these factors (Elmagarmid *et al.*, 2007).

| | **StoreName** | **Activity** | **Street** | **City** | **PostalCode** |
|---|---|---|---|---|---|
| $t_1$ | Mark food | Grocery Meat | 11 Success | Brantford | Z9Z 5Z5 |
| $t_2$ | Rio Store | Vegan | 9 Sun Av | Halifax | X3F 4W5 |
| $t_3$ | Sunny | Naturals | 578 Start St | Calgary | V4T 1E5 |
| $t_4$ | store Mark food | Varieties | 11 Success St | Brantford | Z9Z 5Z5 |
| $t_5$ | Makr food store | Grocery Meat | Success St | Bradford | Z9Z 5Z5 |
| $t_6$ | Mark food store | Grocery Meat Vegan | Succ St | Brantford | Z9Z 5Z5 |
| $t_7$ | Salutare health | Organic | 399 Well Av | Quebec | Y3U 8H4 |

Table 1.1: An example of unclean data in a database

*Data deduplication* process is defined by (Fan and Geerts, 2012) as follows: For a pair of relation schemas ($R_1$,$R_2$); lists $L_1$ and $L_2$ of attributes in $R_1$ and $R_2$, respectively; and an instance ($I_1$,$I_2$) of ($R_1$,$R_2$), data deduplication aims to identify all tuples $t_1 \in I_1$ and $t_2 \in I_2$, given data sources $I_1$ and $I_2$, such that $t_1[L_1]$ and $t_2[L_2]$ refer to the same real-world entity.

The data deduplication process is critical and important to several fields such as data integration, data quality management, and fraud detection (Fan and Geerts, 2012). Data integration systems aim to allow users to access data from multiple heterogeneous data sources through providing a unified view of this data (Bleiholder and Naumann, 2009).

For example, the data in table 1.1 represents a consolidated table that contains information about stores in a central database of an organization that collects data about stores from other databases of different branches worldwide, and saves the data in a central database. The information includes the name of the store, relevant activity, and location including street, city, and postal code. We note that there is a possibility that $t_1$, $t_4$, $t_5$, and $t_6$ tuples may refer to the same store or the same entity, although there is not an exact match between the attribute values of each pair of tuples. In order to clean the data, we need to make decisions about the correct values that represent the final single tuple of $t_1$, $t_4$, $t_5$, and $t_6$.

Duplicate records of a same real-world entity are usually close to each other according to a distance measure, that is, the distances are small, and they have a small number of other neighbors within a small distance (Chaudhuri *et al.*, 2005). We developed a density-based cleaning approach that utilizes the embedded density information in the data to identify a single tuple in the set of tuples pertaining to the same real-world entity. The density information comes from the density-based clustering (Ester *et al.*, 1996) that partitions the tuples into clusters. The main reason to recognize the clusters in this type of clustering is that within each cluster there is a region with a density that is higher than in other scattered, low-density regions. *Density information* refers to the extent of how compact the tuples, i.e. data points, are within a cluster w.r.t. a specified number of neighbors and a distance threshold. Thus, more compactness means there is more confidence in the correctness of the attribute values of the tuples so that confidence scores can be assigned automatically.

For example, in table 1.1, the density information reveals that tuples $t_1$, $t_4$, and $t_6$ lie in a dense region, while tuple $t_5$ lies in a lower density region. We utilize this fact in the deduplication process. We consider, basically, that the weights of tuples $t_1$, $t_4$, and $t_6$ are higher than that of tuple $t_5$. Then we use the assigned weights to guide the cleaning process further to fix quality issues.

This approach meets the objectives of manual user interaction-independent data cleaning, master data-independent data cleaning, and training data-independent data cleaning.

## 1.2 Repairing Violations of Functional Dependencies

Integrity constraints such as functional dependencies (Abiteboul *et al.*, 1995) could be violated for different reasons such as integration with other data sources. This violation makes data inconsistent, which in turn makes the data unclean. For example, consider table 1.1 and the functional dependency $F : [PostalCode] \rightarrow [City]$, that is, any two tuples that have equal values in the attribute **PostalCode**, must have equal values in the attribute **City**. The attribute values of the city in $t_1$, $t_4$, $t_5$, and $t_6$ violate the FD because there are two values, "Brantford" and "Bradford" for the postal code "Z9Z 5Z5". Different approaches have been introduced to repair the data in the table. One approach (Chomicki and Marcinkowski, 2005) repairs the data by deleting the tuples with errors. To repair the data in **City**, we may delete tuple $t_5$. Another approach (Bohannon *et al.*, 2005) repairs the data by modifying the attribute values. For example, a possible repair is to modify $t_5$[City] to "Brantford" value. Another possible repair is to modify the values of $t_1$[City], $t_4$[City], and $t_6$[City] to "Bradford". The first repair might be preferred as it requires fewer modifications. The approach of value modification is preferred over tuple deletion as there is more information lost in the tuple deletion approach.

However, there are cases in which there is no certainty about which values to repair. For example, let us assume that there are the values "Brantford", "Brantfor", "Brandord", and "Branftord" in $t_1$[City], $t_4$[City], $t_5$[City], and $t_6$[City], respectively, instead of the values shown in table 1.1. The attribute values of the city in $t_1$, $t_4$, $t_5$, and $t_6$ violate the FD because there are four values for the postal code "Z9Z 5Z5".

Among the four values, “Brantford”, “Brantfor”, “Brandord”, and “Branftord”, the last three values do not seem to be correct as names of cities. This could be caused by transcription errors. Minimum change may not be enough as there are three possible changes for each repair. Recall in the previous section we utilized the density information to assign weights. In a case such as this we may incorporate weights of tuples in the decisions of repairing.

We developed a density-based cost model that utilizes the weights that are based on the density information. This repair results in minimal change to each set of duplicate tuples. We incorporate the weights of tuples to decide which inconsistent attribute value to repair w.r.t. a set of functional dependencies. This method assumes that in a database, the clean part is larger than the unclean part, and thus only a small number of changes to the attribute values would make the data satisfy the FDs. Thus, among the four values in **City**, we could conclude that the value “Brantford” has the lowest cost to modify other values into and hence, it is the correct value.

This approach meets the objectives: of fixing duplicate records issue and inconsistent data quality issue caused by violations of FDs; of handling a variety of error causes; and of manual user interaction-independent data cleaning. In this thesis, we consider the inconsistent data in terms of violations with respect to a set of functional dependencies (FDs), as their violations are common in practice.

## 1.3 Fixing Inaccurate and Outdated Data

Identifying a single tuple in the duplicate elimination process may require more than just repairing integrity constraints violations, as there are other data quality issues. For example, in table 1.1, some of the attribute values in tuples have some differences that cannot be resolved by repairing the violations of integrity constraints because there are no integrity constraints defined over these attributes. In $t_4$[StoreName], the value is "store Mark food", while in $t_6$[StoreName], the value is "Mark food store". This may happen when there is no standard format. In this case, there is no stipulated criterion for how to write a business name. In $t_5$[StoreName], part of the name is the value "Makr". The last two adjacent characters are transposed. This could be a transcription error. In **Street**, the correct street value is also not clear. It could be, for example, "Success St", "11 Success St", or "11 Success". We need to make decisions about all these different values.

In the attribute **Activity**, assuming that we know that "Vegan" has been recently added, and that an activity that contains "Vegan" is most current, we may conclude that $t_6$[Activity] is more recent than $t_1$[Activity] and $t_5$[Activity] and therefore most accurate. However, it is still not possible to identify a final single attribute value in **Activity**. This is because the value in $t_4$[Activity] is different. We cannot identify it just by using the facts that are related to "Vegan" food. The currency information does not provide sufficient evidence to make a decision about which value to support in **Activity**. In addition, there are no timestamps in the data that may help to support one value over another in this case.

For the class of the outdated values that have neither complete currency information nor timestamps, we developed techniques and algorithm to discover the accurate and current values. For example, to make decisions about which value is most current between $t_4$[Activity] and $t_6$[Activity], we use the density information to corroborate which value is more trustworthy. Thus, we could curtail that $t_6$[Activity] as most current.

This approach meets the objectives: of fixing duplicate records and outdated data issues; of handling a variety of error causes; and of manual user interaction-independent data cleaning.

For the class of inaccurate values that do not have integrity constraints defined over them, but have errors introduced by transcription and a lack of standard formats, we developed techniques and algorithms to fix these issues. The first technique is based on the agglomerative clustering, by measuring the relatedness of the words of the attributes in the tuples. Agglomerative clustering starts with singleton clusters of attribute values and successively combines pairs of clusters that are the closest until the desired cluster structure is obtained. This functionality is incorporated in the deduplication process to handle the inaccuracy in attribute values. For example, by merging the attribute values in **Street**, we could conclude that "11 Success St" is the accurate value. Second, we developed a technique that is based on data density to corroborate the accuracy of the attribute values in the data. By utilizing the density information, we corroborate the trustworthiness of the attribute values to fix the inaccuracy. This technique also incorporates the currency information, if any. For example, based on the density of data, and the fact that tuple $t_6$ has another current value, which is $t_6$[Activity], we could conclude with higher confidence that the value $t_6$[StoreName] is more accurate than other values.

This approach meets the objectives: of fixing duplicate records and inaccurate data issues; of handling a variety of causes of errors; and of manual user interaction-independent data cleaning.

## 1.4 Contributions and Thesis Outline

In this thesis, we provide a data cleaning approach that leverages density-based information during the data cleaning process, with neither manual user-interaction nor dependency on master data or training data. We apply our approach in terms of four data quality issues: duplicate tuples, inaccurate data, inconsistent data, and outdated data.

We present in Chapter 3 our algorithms for deduplication with inconsistent data repairing and discovering of the accurate values in the data. We introduce a weight model that utilizes the density information in the data to assign weight scores. We present a cost model that utilizes the weights to select the best repair. Toward the goal of fixing multiple quality issues, we incorporate hierarchical clustering to discover the accurate values, resulting in a framework that has the functionality of cleaning duplicate tuples that have violations of FDs and inaccurate values. Finally, we evaluate the effectiveness of our techniques through conducting qualitative and performance evaluations, and we use synthetic and real datasets. Some preliminary results have been accepted for publication in SAI 2016 (Al-janabi and Janicki, 2016b).

In Chapter 5, we present our algorithms of data cleaning that determine what values are more accurate and more current during the data deduplication process. We introduce a confidence model that utilizes the currency order and currency constraints to assign confidence score to tuples. We employ techniques to corroborate the attribute values in order to determine the current and accurate values. Finally, we conduct qualitative and performance evaluations to show the effectiveness of our techniques. Some preliminary results have been accepted for publication in IntelliSys 2016 (Al-janabi and Janicki, 2016a).

This thesis is organized as follows: In Chapter 2, we give a background and review of the research areas related to our focus areas herein. In Chapter 3, we present our density-based approach for deduplication with consistency and accuracy, and then in Chapter 4, we experimentally evaluate the quality and performance of the approach. In Chapter 5, we present our density-based corroborating approach for deduplication with data currency and accuracy, and then we experimentally evaluate the quality and performance of the approach in Chapter 6. In Chapter 7, we present directions for future work and in Chapter 8, we present conclusions and final comments.

# Chapter 2

# Background

In this Chapter, we discuss the relevant background material to our research topics. In sections 2.1 and 2.2, we introduce the relational model and data cleaning, respectively. In section 2.3, we discuss data clustering. In sections 2.4 through 2.7, we discuss several prominent data quality issues that we focus on in this thesis including duplicate records, integrity constraints violations, outdated data, and inaccurate data. Related work that is specific to each Chapter is presented in the corresponding Chapter.

## 2.1 The Relational Data Model

In the relational model, data is stored in relations and databases are represented as a collection of *relations*. A relation is also known as a table. A relation contains rows, where each row represents a relationship among a set of data values. The row is also called *tuple*. A real-world entity is represented by each tuple. A relation also contains columns, where each column header is called an *attribute*.

*Truck*

| | **Number** | **Make** | **Model** | **EngineSize** | **ManYear** | **TowingCapacityPnd** |
|---|---|---|---|---|---|---|
| $t_1$ | 35246 | MACS | AB99 | 3.7 | 2011 | 3500 |
| $t_2$ | 11949 | FEDO | SC300 | 4.3 | 2012 | 4550 |
| $t_3$ | 24570 | LEMI | KLG3 | 5.7 | 2012 | 9200 |
| $t_4$ | 14449 | FEDO | SC300 | 4.3 | 2010 | 3800 |
| $t_5$ | 20540 | LEMI | KLG3 | 5.7 | 2015 | 7300 |
| $t_6$ | 67124 | LEMI | KLG3 | 5.7 | 2015 | 9200 |
| $t_7$ | 98780 | STON | LZK1 | 6.0 | 2011 | 10250 |
| $t_8$ | 35246 | MACS | AB99 | 3.7 | 2010 | 3500 |
| $t_9$ | 95244 | STON | RMN2 | 6.6 | 2015 | 13000 |
| $t_{10}$ | 95297 | STON | RMN2 | 6.6 | 2015 | 13000 |

Table 2.1: Example of inclusion dependency

The data type associated with the attribute values of each column is called the attribute *domain*. A specific set of rows of a relation is called *relation instance*. A relational database typically has many relations.

Table 2.1 is an example of a relation called *Truck* that contains data about trucks. The relation consists of ten tuples. It has also six attributes: truck number (**Number**), make (**Make**), model (**Model**), engine size (**EngineSize**), manufacturing year (**ManYear**), and towing capacity in pounds (**TowingCapacityPnd**). An example of a data type that is associated with the domain of **EngineSize** can be declared as a real number between 1 and 7.

## 2.2 Data Cleaning

There are several causes of data quality degradation. Some of these causes include existence of duplicate records (Elmagarmid *et al.*, 2007; Mulry *et al.*, 2006; Winkler, 2006), violations of integrity constraints (Cong *et al.*, 2007; Bohannon *et al.*, 2005; Kolahi and Lakshmanan, 2009), missing data (Fan and Geerts, 2010; Müller and

Freytag, 2003), outdated data (Chomicki and Toman, 2005), and inaccurate data (Batini and Scannapieca, 2006). Data cleaning process aims to fix the problems in data quality aspects.

## 2.3 Data Clustering

Broadly, *data clustering* can be defined as a method to partition a dataset $D$ that has a set of objects $O$ in a space $S$, into $g$ meaningful groups (i.e. clusters) such that the objects in each group are similar to one another as compared to other objects in other groups, according to some similarity criteria. Clustering is also known as unsupervised learning in a machine learning field. Data clustering has been studied in different areas such as machine learning (Witten *et al.*, 2011), statistics (Xu and Wunsch, 2005), data mining (Larsen and Aone, 1999; Witten *et al.*, 2011), and duplicate elimination (Elmagarmid *et al.*, 2007).

Clustering algorithms have differences in their properties. For example, some algorithms provide a representative for each group. Some of them assign each group a distinct group of objects (i.e. hard clustering), while other clustering algorithms follow soft clustering methods in which objects in different clusters may overlap, with probability distributions over the membership of the cluster. Some algorithms are able to isolate outliers that do not belong to any group (refer to (Xu and Wunsch, 2005) for a survey).

### 2.3.1 Similarity Measures

The similarity measure operation is used to group similar data objects in data clustering, and it is a key factor in how a clustering algorithm is able to successfully find objects that are similar to each other, and keeps dissimilar objects away. That is, to measure the closeness between objects. Distance measures are an alternative concept that measure the dissimilarity between objects. There is a large number of (dis)similarity measuring methods that have been proposed in the literature. We review some of the measures in this subsection.

A class of similarity distances is known as the family of *Minkowski distances* (Cios *et al.*, 1998). Given two vectors **x** and **y** that represent two data points in $d$-dimensional space, Minkowski distance of order p is given by:

$$||\mathbf{x} - \mathbf{y}||_p = (\sum_{i=1}^{d} |x_i - y_i|^p)^{1/p} \tag{2.1}$$

where $\mathbf{x}$,$\mathbf{y} \in \mathbb{R}^p$. For $p = 2$, the distance is called *Euclidean distance*, and defined as:

$$||\mathbf{x} - \mathbf{y}||_2 = (\sum_{i=1}^{d} |x_i - y_i|^2)^{1/2} \tag{2.2}$$

For $p = 1$, the distance is called *Manhattan distance*, and defined as:

$$||\mathbf{x} - \mathbf{y}||_1 = \sum_{i=1}^{d} |x_i - y_i| \tag{2.3}$$

*Cosine similarity* measure is another similarity measure in the field of information retrieval. It computes the cosine of the angle between the corresponding vectors, and defined as:

$$sim_{cos}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{||\mathbf{x}|| \, ||\mathbf{y}||} \tag{2.4}$$

where $||\mathbf{x}||$ indicates the length of the vector $\mathbf{x}$ (resp. $\mathbf{y}$). There are other types of measures. For example, two measures that handle character-based errors are *Levenshtein distance* (Levenshtein, 1965) and *q-grams* (Ukkonen, 1992) (refer to (Elmagarmid *et al.*, 2007) for a survey). In the *Levenshtein distance* measure, the distance between two strings, $s_1$ and $s_2$, is the minimum number of edits needed to transform $s_1$ into $s_2$. The types of edit operations are insertions, substitutions, or deletions. For example, consider two strings $s_1$ = "Bean" and $s_2$ = "Beens". Levenshtein distance $LD(s_1,\ s_2)$ between $s_1$ and $s_2$ is 2, as the 'a' in $s_1$ needs to be replaced by an 'e', and an 's' needs to be inserted in $s_1$. In the *q-grams* similarity measure, a string is divided into smaller tokens of size q. For two strings $s_1$ and $s_2$ to be similar, they should share a large number of q-grams in common. Many clustering algorithms use a (dis)similarity matrix that contains the pairwise (dis)similarities between the objects being considered.

### 2.3.2 Categorization of Data Clustering Algorithms

There are different clustering algorithms that have been proposed due to different types of application domains (Jain and Dubes, 1988). Properties such as clustering criteria and starting points can be used to differentiate clustering algorithms. We describe in this subsection some of the main properties (Jain *et al.*, 1999).

**Agglomerative vs. Divisive Clustering**: *Agglomerative clustering* (bottom-up hierarchical clustering) starts with each object in a singleton cluster; meaning that each object is in its own cluster. Then it starts merging successively into larger clusters until the desired cluster structure has been obtained. *Divisive clustering* (top-down hierarchical clustering) starts with all the objects in one cluster, and then it performs splitting until the desired cluster structure has been obtained.

**Hard vs. Fuzzy Clustering**: *Hard clustering* algorithms allocate each object to one and only one cluster. In *Fuzzy clustering*, an object is assigned a probability of being a member of a cluster. The higher the probability of an object belonging to a cluster, the higher the confidence that the object belongs to that cluster.

**Polythetic vs. Monothetic Clustering**: *Polythetic clustering* uses all the features at once during clustering; that is, similarities are measured by a simultaneous use of features. In *Monothetic clustering*, the features are used one by one. Most clustering algorithms are polythetic.

**Incremental vs. Non-Incremental Clustering**: *Incremental clustering* methods are used in the case that there is a large set of objects to be clustered, and constraints on memory space or execution time may affect the design of the clustering algorithm. Algorithms of incremental clustering minimize the number of scans through the set, reduce the size of data structures used in the operations of the algorithm, or reduce number of objects examined during execution.
Others (Januzaj *et al.*, 2003) distinguish further properties such as:

**Distance vs. Density Clustering**: In *Distance-based clustering*, an object is assigned to a cluster based on its distance from, for example, the representative of the cluster. The distances between objects are used directly to optimize a global criterion function. In *density-based clustering*, a cluster is a region of high density surrounded by a region of lower density. A local cluster criterion is applied in which an object is assigned to a cluster as long as the number of neighborhood data points satisfies a threshold. It is an efficient method when outliers are presented, or when the clusters are irregular or intertwined.

**Hierarchical vs. Partitional Clustering**: *Hierarchical clustering* seeks to build a hierarchy of clusters where clusters merge or split depending on the type of the hierarchical clustering at certain distances. The results are presented in a tree structure called a dendrogram. Starting from an initial partition, *Partitional clustering* methods, on the other hand, relocate data points by moving them from one cluster to another. They decompose the data into a set of disjoint clusters that reflect the natural groups presented in the data, as opposed to, for example, the dendrogram. An advantage of hierarchical clustering is that it does not require a number of clusters a priori. A drawback in partitional clustering is that it does require a number of clusters to be specified a priori.

### 2.3.3 Data Clustering Algorithms

We present in this subsection some of the unsupervised clustering algorithms including agglomerative hierarchical, DBSCAN, K-means, and Mean-shift algorithms.

#### Agglomerative Hierarchical Clustering

The agglomerative hierarchical algorithm starts with assigning individual data point as clusters, and then it computes the (dis)similarity matrix between pairs of clusters. After that, it looks for a pair of clusters with the shortest distance to merge. The merge of the two closest clusters continues repeatedly until only one cluster remains. Algorithm 2.3.3.1 describes the basic steps of this type of clustering.

**Algorithm 2.3.3.1:** Agglomerative Hierarchical Clustering

```
1: Create a collection $\mathbf{G}$ of $g$ singleton clusters
2: Assign each cluster one data point $o$
3: Compute the (dis)similarity matrix
4: repeat
5:   Find the closest pair of clusters: $dist(c_n, c_m)$
6:   Merge $c_n$, $c_m$ into a new cluster $c_{n+m}$
7:   Remove $c_n$, $c_m$ from $\mathbf{G}$ and add $c_{n+m}$ to $\mathbf{G}$
8:   Update the similarity matrix
9: until Only one cluster remains
```

In addition to the similarity measure, a merging approach needs to be specified to decide how to merge clusters and this is known as a linkage criterion. We discuss in this subsection some linkage criteria.

**Single Linkage**: This type of linkage considers the distance between two clusters to be equal to the minimum distance (i.e. the lowest dissimilarity in the case of dissimilarity matrix) from any member of one cluster to any member of the second cluster. This method has a drawback known as *chaining effect* when two distinct clusters may be joined together simply because just one of the members in the first one is close to another member of the second one.

**Complete Linkage**: This type of linkage considers the distance between two clusters to be equal to the furthest distance from any member of one cluster to any member of the second cluster. It solves the problem of chaining, however, it may create another one in which two close clusters may be joined together later, in the case they contain one remote pair of members.

**Average Linkage**: This type of linkage considers the distance between two clusters to be equal to the average pairwise distances between all the pairs in the clusters. It is considered as an intermediate approach between single linkage and complete linkage.

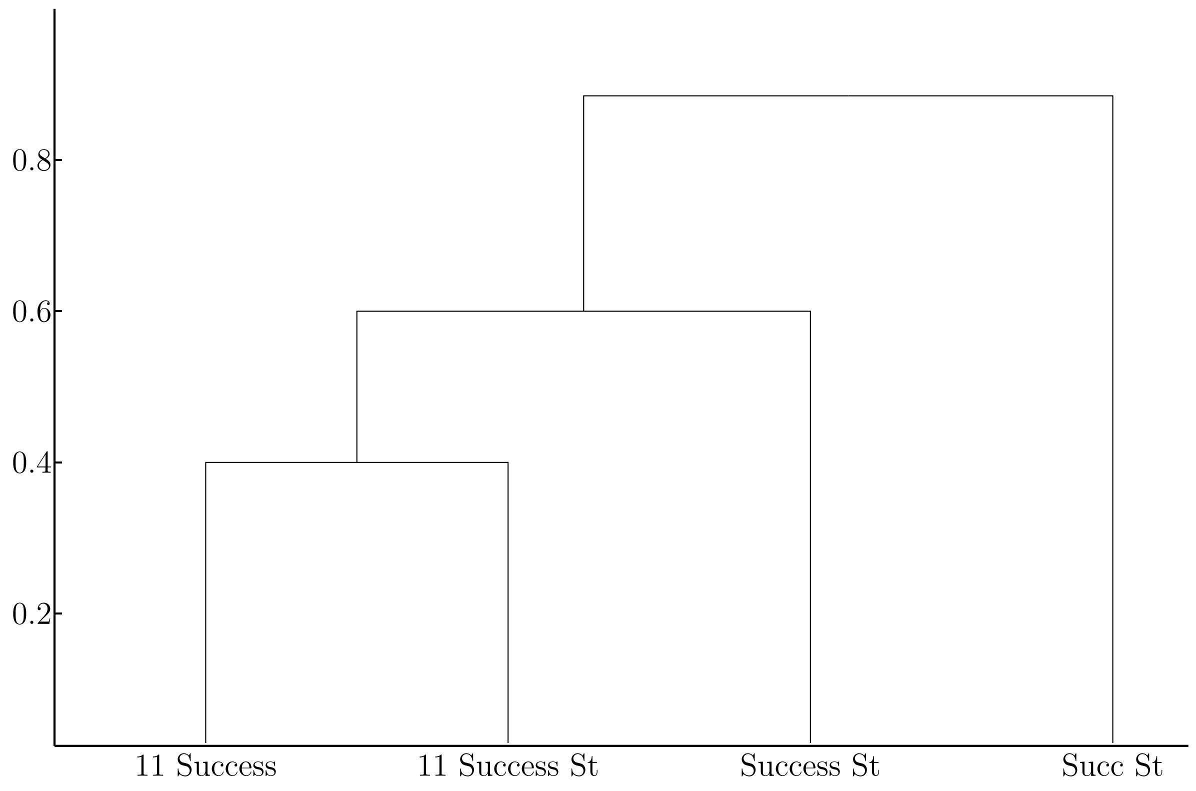

Figure 2.1: Dendrogram of the data of an attribute in table 1.1

Figure 2.1 illustrates the results represented as a dendrogram of an agglomerative clustering for the values of the attribute **Street** in table 1.1. The clustering started with four singleton clusters: “11 Success”, “11 Success St”, “Success St”, and “Succ St”. At a dissimilarity value of 0.4, the first two clusters are merged to form a fifth larger cluster. At a dissimilarity value of 0.6, the third singleton cluster is merged with the fifth cluster to form a sixth larger cluster. At a dissimilarity value of 0.885, the fourth singleton cluster is merged with the sixth cluster. The last resultant cluster contains all the singleton clusters. It is possible to cut the hierarchy at some dissimilarity value. For example, we cut at 0.5 if we want clusters with a combination dissimilarity of 0.5. This yields only the first two merged singleton clusters being the resultant final cluster, that is, “11 Success” and “11 Success St” are merged into one cluster. In our technique, we get the most accurate value in **Street** by eliminating the repeated words from the resultant cluster (i.e. ”11” and ”Success”), which yields “11 Success St” as the most accurate value.

### DBSCAN Algorithm

We describe DBSCAN (Ester *et al.*, 1996), a density-based data clustering algorithm in this section. It uses a density threshold around each object in order to differentiate the relevant data point from outliers. Density is estimated for a particular data point through counting the number of data points, including the data point itself, within a specified radius. The neighborhood of a data point is a set of all data points that have a distance measure within a specified radius (i.e., the cardinality of the neighborhood $\geq$ density threshold).

Data points in the data can be divided into three types: 1) core points in the interior of the dense region, 2) border points located on the extremities of the dense region, or 3) data points in a sparsely dense region (outliers). A cluster contains core points as well as border points.

Generally, the neighborhood size of the core points is bigger than that of the border points. Based on the notion of density-based reachability, a cluster can be defined as the maximal set of reachable core points. That is, any two core points that are close to each other w.r.t. the radius are put with each other in the cluster. Any border point that is close to a core point would be put in the same cluster.

**Algorithm 2.3.3.2:** DBSCAN

```
1: for each data point $o \in$ data do
2:     if $o$ is not yet clustered then
3:         Retrieve all data points which are density-reachable from $o$
           w.r.t. the radius and density threshold
4:         if $o$ is core tuple then
5:             Form a cluster $C$ w.r.t. the radius and density threshold
6:             Mark data points of $C$
7:         else
8:             Mark $o$ as noise
```

DBSCAN is described in Algorithm 2.3.3.2. It selects a data point that has not been clustered and retrieves its density-reachable tuples w.r.t. radius and density threshold. In the case the selected tuple is core, a cluster is formed. The process continues until all the data points that belong to this cluster, including core and border data points, are retrieved. The data points that belong to a cluster are marked as belonging to this cluster. In the case that the selected data point is not core, it is marked as a noise (i.e., an outlier). This outlier might be found later to be a part of another cluster.

Figure 2.2 illustrates an example of DBSCAN clustering of some dataset. There are three clusters where the membership of data points is represented by the blue, green, and cyan colors. The outliers are colored in red. The border data points are visualized with smaller circles than that of core points.

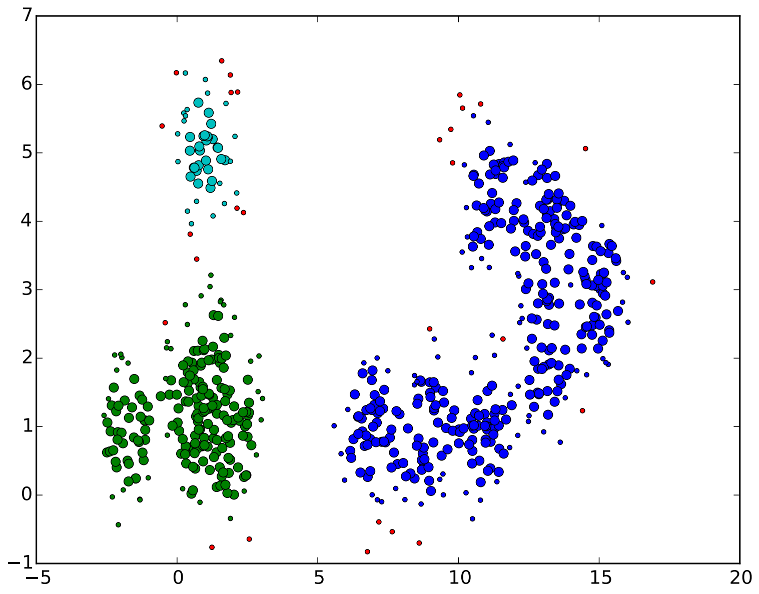

Figure 2.2: DBSCAN clustering

### K-means Algorithm

The K-means algorithm (MacQueen, 1967) is a partitional-based algorithm that is based on the idea that a center data point can be used to represent a cluster. The K-means algorithm defines a centroid, which is usually the mean or the median point of a group of data points. The centroid may not correspond to an actual data point. The shapes of the clusters in K-means have convex shapes. It is usually used with numeric data types. K-means requires a number of clusters $K$ to be specified a priori, and it is sensitive to outliers.

**Algorithm 2.3.3.3:** K-means

```
1: Initialize $K$ data points as initial cluster centroids
2: repeat
3:     Compute the distance of each data point to each cluster centroid
4:     Assign each data point to the cluster with the closest cluster centroid
5:     Recompute the centroid of each cluster
6: until Centroids no longer change
```

Algorithm 2.3.3.3 describes the basic K-means clustering. It starts by assigning $K$ data points as initial centroids. Then, repeatedly, for each data point, the algorithm computes the distance from each data point to each cluster centroid. Each data point is assigned to the closest cluster centroid. The repetition process continues until no points change cluster memberships when recomputing the centroid of each cluster.

Figure 2.3 illustrates K-means clustering for the same dataset that is used in DBSCAN clustering in figure 2.2. The value of $K$, that is, the number of clusters, is 3. Figure 2.4 shows K-means clustering for the same dataset but when $K$ is 4. Each cluster is colored differently.

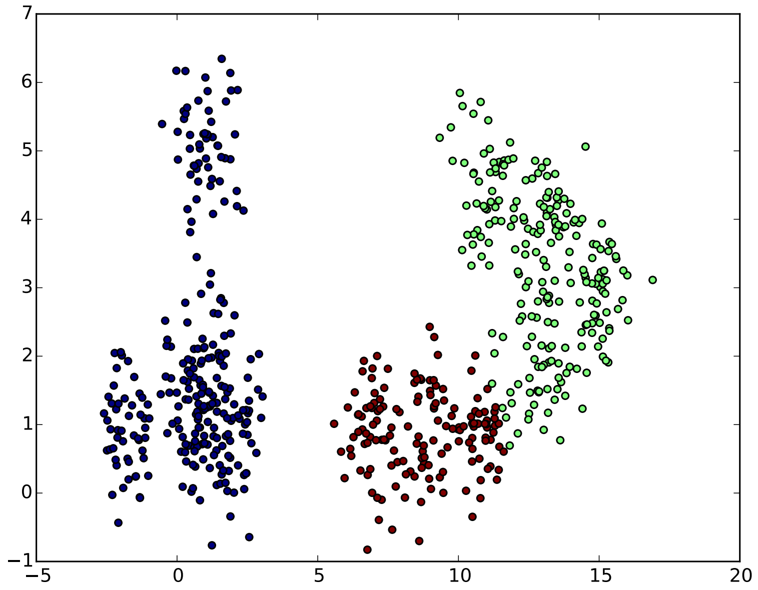

Figure 2.3: K-means clustering with $K$=3

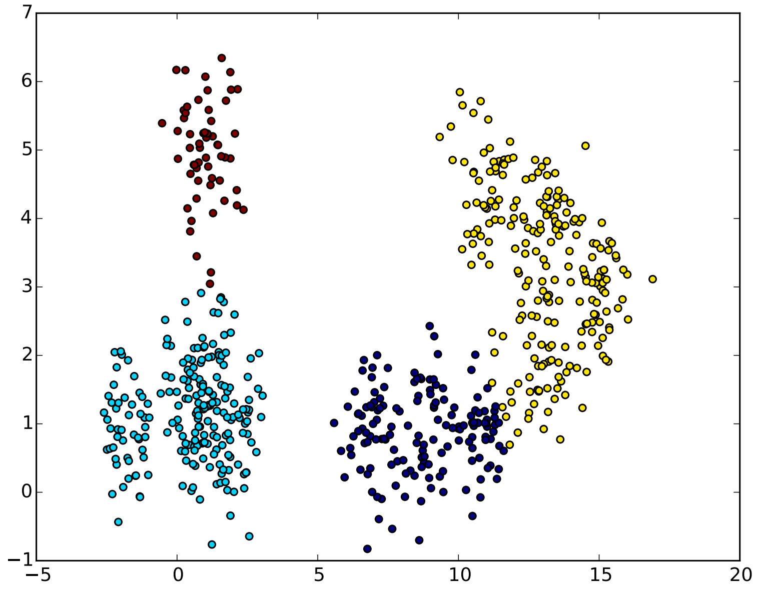

Figure 2.4: K-means clustering with $K$=4

#### Mean-shift Algorithm

Mean-shift algorithm (Fukunaga and Hostetler, 1975; Cheng, 1995; Comaniciu and Meer, 1999, 2002) is a density-based algorithm that iteratively shifts each data point to the mean of data points in its neighborhood. It locates the maxima of a density function in order to find the dense regions. Mean-shift does not need a number of clusters a priori and can handle arbitrary shapes of clusters. However, there is more computational cost in the mean-shift algorithm compared to DBSCAN. Also, it does not scale well with large datasets.

Algorithm 2.3.3.4 describes the basic mean-shift clustering. Mean-shift defines a window around each data point and then it computes the mean of the density in the window according to the density function. After that, the window is shifted to the new mean, that is, more dense region. These steps continue repeatedly until convergence.

**Algorithm 2.3.3.4:** Mean-shift

```
1: repeat
2:     Define a window around each data point
3:     Compute the mean of data points within the window
4:     Shift the window to the mean
5: until Convergence
```

Figure 2.5 illustrates mean-shift clustering for the same dataset that is used in DBSCAN clustering in figure 2.2. Each cluster is colored differently.

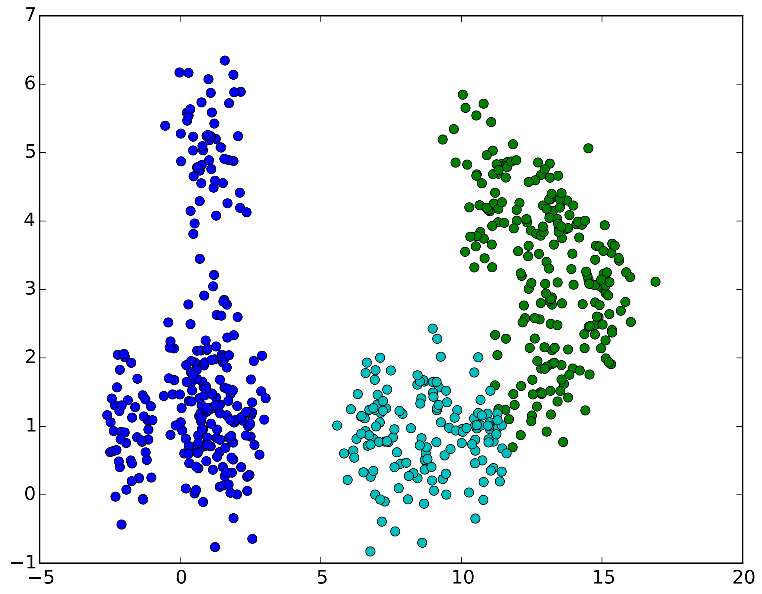

Figure 2.5: Mean-shift clustering

## 2.4 Duplicate Records

Data deduplication is also known by other terms such as duplicate detection, merge-purge, record linkage, duplicate elimination, and record matching. For example, the data in figure 2.6 represent two databases of information about earthquakes that are to be integrated. The *Seismic* database contains information about earthquakes that happened worldwide. The *Earthquake* database contains information gathered from earthquake detection stations in South America.

*Seismic*

| | **EQID** | **Date** | **Mag** | **Lat** | **Long** | **Depth** | **Location** |
|---|---|---|---|---|---|---|---|
| $t_1$ | 23529865 | 17/5/14 5:44 PM | 5.2 | -24.1 | -65.4 | 95.7 | Argentina, S. America |
| $t_2$ | 23527864 | 17/5/14 5:12 PM | 2.1 | 19.7 | -165.2 | 210.4 | Pacific Ocean |
| $t_3$ | 23527863 | 17/5/14 4:34 PM | 2.1 | 19.7 | -6.9 | 129.8 | Banda Sea |

*Earthquake*

| | **StationNumber** | **Magnit** | **Latit** | **Long** | **Date** | **Time** | **Area** |
|---|---|---|---|---|---|---|---|
| $t_1$ | 124 | 4.0 | -19.41 | -70.02 | 17/05/2014 | 21:01:48 | Chile |
| $t_2$ | 123 | 4.8 | -8.83 | -74.81 | 17/05/2014 | 20:34:23 | Peru |
| $t_3$ | 122 | 5.2 | -24.09 | -65.40 | 17/05/2014 | 17:44:24 | Argentina |

Figure 2.6: Duplicate records resulting from data integration

The **EQID** attribute in the *Seismic* represents the ID of an earthquake. Other information includes: the magnitude of the earthquake, the longitude and latitude of the area where the earthquake occurred, the depth of the earthquake, and the location of where the earthquake happened. The attribute **StationNumber** in the *Earthquake* represents the station number. If the case data are to be merged from these two databases into one consolidated table, tuple $t_1$ in *Seismic* and tuple $t_3$ in *Earthquake* could refer to the same earthquake, which introduces duplicate records.

Several systems (e.g., (Galhardas *et al.*, 2001; Le *et al.*, 2000)) divide the deduplication process into three steps: record or tuple matching, partitioning the tuples into disjoint clusters (i.e. clustering), and merging the tuples of each cluster into a single tuple.

In the first step, distance measure or similarity measure is usually used to classify a candidate pair as duplicate or non-duplicate. Field matching techniques are used to find duplicate fields. Different kinds of distance measures can be used, such as, character-based metrics which are designed to handle typographical errors, such as, Edit distance and Jaro distance. Other types of metrics could be used, including, token-based metrics and numerical values metrics. Examples of such metrics are Jaccard distance, Q-grams with tf.idf, and Minkowski. In reality, records have multiple fields, so these need to be matched (Elmagarmid *et al.*, 2007).

A weighted undirected graph can be used to represent the output of the tuple matching step, where the nodes represent the set of tuples, and each edge connects two similar tuples according to a pairwise classification. The weight represents the similarity between the edge nodes. Since pairwise comparison between all tuples is expensive due to large volume of input data to be deduplicated, optimization techniques are used to get around this problem. An optimization method introduced by (McCallum *et al.*, 2000) divides tuples into overlapping clusters called *canopies*, using an inexpensive measure to compute similarity. Then, a more expensive similarity measure is used. By using the inexpensive similarity measure first, the computations of pairwise comparisons are saved by just comparing among tuples within a common canopy. Another optimization method (Hernández and Stolfo, 1998) called *Sorted-Neighborhood* consists of three phases. First, each record is assigned a sorting key. Second, the records are sorted based on the sorting key. And finally, a window of size $m$ is moved sequentially over the sorted records where the records within the same window are compared. The use of windows limits the comparisons for matching records, and sorting records keeps similar records close to each other.

Table 2.2, for example, represents data about books including title of book and its volume. Obviously, tuples $t_1$ and $t_2$ are duplicates. Similarity-based duplicate classification between the tuples might detect the following duplicate pairs: ($t_1$, $t_2$) and ($t_2$, $t_3$). Figure 2.7 represents the graph of the classification.

| | **Author** | **Title** | **Volume** |
|---|---|---|---|
| $t_1$ | Mike Stewart | Programming with Java | 1 |
| $t_2$ | M. Stewart | Programming with Java | V.1 |
| $t_3$ | Marsel Steven | Programming with Java | V.1 |
| $t_4$ | Jack Joseph | Data Management Concepts | 2 |

Table 2.2: Sample of books data

In the second step, a clustering algorithm partitions the tuples such that each cluster represents a different real-world entity. Clustering is necessary as there is a drawback in the first step in which the matched tuples may not represent a transitive relation. For example, in figure 2.7, tuple $t_1$ is considered to be similar to tuple $t_2$, and tuple $t_2$ is considered to be similar to tuple $t_3$, but tuple $t_1$ is not similar to tuple $t_3$. In fact, each of $t_1$ and $t_3$ refer to a different real-world entity. A clustering algorithm aims generally to maximize the intra-cluster similarity (high intra-cluster similarity) and minimize inter-cluster similarity (low inter-cluster similarity) (Chen *et al.*, 1996), that is, tuples inside a cluster are similar to one another, but differ from tuples in other clusters.

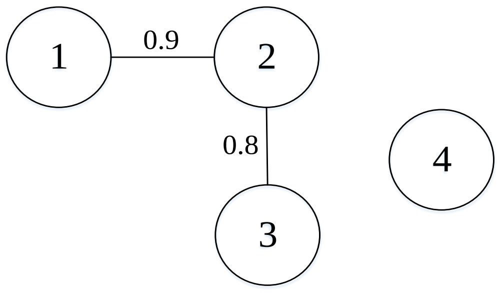

Figure 2.7: Graph representation of similarity-based duplicate classification

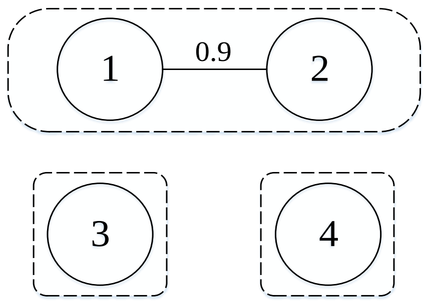

Figure 2.8: Clusters after edge removal

One approach is based on the graph of the classification (Naumann and Herschel, 2010). It tackles this problem by removing recursively the edges that do not fall within the lowest similarity, according to the increasing order of the similarity, until some adequate result is obtained. For example, in figure 2.7, we observe that the similarity between $t_2$ and $t_3$ is low, so the edge is removed so that $t_2$ and $t_3$ fall into different clusters. Figure 2.8 depicts the final result in which there are three clusters.

Another clustering approach (Chaudhuri *et al.*, 2005) is based on two criteria: compact set and the sparse neighborhood. Compact set represents the intuition that duplicate tuples are closer to each other, as compared to their closeness to other distinct tuples. That is, for a threshold $k$, duplicate tuples should be $k$-nearest neighbors of each other. Sparse neighborhood represents the intuition that the local neighborhood of duplicate tuples is sparse, that is, not many tuples are in the area surrounding the duplicate tuples.

The final step of deduplication is to merge the tuples or records of each cluster into one single representative tuple or record. In one approach (Galhardas *et al.*, 2001), for example, the tuples in each cluster are merged into a single tuple using some aggregate function. This aggregate function could be user-defined.

## 2.5 Integrity Constraints

The tuples in a relational database are usually related to each other. Generally, there are constraints on the actual values that can be stored in a database. Real-world applications normally have specific semantics and requirements that the data in databases must satisfy. Integrity constraints are widely used to incorporate semantics to the data in a relational database. However, these constraints could be violated for a variety of reasons including integration with other data sources. This violation makes data inconsistent which means it does not represent the correct semantics which makes the data unclean.

There are several kinds of prominent integrity constraints including functional dependencies, multivalued dependencies, join dependencies, inclusion dependencies (Abiteboul *et al.*, 1995), conditional inclusion dependencies (Bravo *et al.*, 2007; Fan and Geerts, 2012), conditional functional dependencies (Bohannon *et al.*, 2007; Fan and Geerts, 2012), and matching dependencies (Fan, 2008). In the following subsections, we give an overview of several types of integrity constraints.

### 2.5.1 Functional Dependencies

Functional dependency (FD) is a widely used form of integrity constraints. It is a property of the semantics of the attributes, that is, a functional dependency (FD) $F : X \rightarrow Y$ defined over a relation $R$, where $X$ and $Y$ are sets of attributes in $R$, states that in any two tuples $t'$ and $t''$ in FD, if $t'[X] = t''[X]$ then $t'[Y] = t''[Y]$. The values of a set of attributes $X$ in a tuple uniquely or functionally determine the values of another set of attributes $Y$ of the tuple. We let $I$ be an instance of $R$. For a set of functional dependencies $\Sigma$, $I \models \Sigma$ ($I$ satisfies $\Sigma$) iff none of the tuples in $I$

violate any F in $\Sigma$.

For example, in table 2.1, the functional dependency $F : [\mathit{Make}, \mathit{Model}] \rightarrow [\mathit{EngineSize}]$ means that the **Make** and the **Model** of the *Truck* relation uniquely determine the **Engine Size** of the truck. One of the main uses of functional dependencies is to enhance the quality of schema design.

### 2.5.2 Conditional Functional Dependencies

Conditional functional dependencies (CFDs) are used to bind semantically related data values. CFDs are defined over a subset of tuples which match some pattern rather than the entire relation, like the case in the functional dependencies. Given a relational schema $R$, a CFD $\phi$ can be defined on $R$ as a pair $(X \rightarrow Y, T_p)$ where $X$ and $Y$ are sets of attributes in $R$, $X \rightarrow Y$ is a functional dependency FD, and $T_p$, referred to as a pattern tableau of $\phi$, is a set of tuples in which its attributes are $X$ and $Y$. The value $t_p[C]$, of an attribute $C \in (X \cup Y)$ for a pattern tuple $t_p \in T_p$, is either constant value in the domain of $C$ or an unnamed variable. If the right hand side of the pattern tuple is '_' then the CFD is called a variable CFD, whereas if the pattern tuple $t_p$ consists of constants only, then the CFD is called a constant CFD. For each attribute $C \in (X \cup Y)$ and for each tuple $t_p \in T_p$, either $t_p[C] =$'_' or $t_p[C] = t[C] =$ some constant in the domain of $C$. An instance $I$ of $R$ satisfies a CFD $\phi(X \rightarrow Y, T_p)$, written as $I \models \phi$, if for each pair of tuples $t_1$, $t_2$ in $I$, and for each pattern tuple $t_p$ in $T_p$ of $\phi$, if $t_1[X]$ and $t_2[X]$ are equal, i.e. $t_1[X] = t_2[X]$, and both match the pattern $t_p[X]$ (written as $t \asymp t_p$) then $t_1[Y]$ and $t_1[Y]$ should be equal and both match $t_p[Y]$. That is, if $t_1[X] = t_2[X] \asymp t_p[X]$ then $t_1[Y] = t_2[Y] \asymp t_p[Y]$.

$\phi_1$: [*Model*] $\rightarrow$ [*EngineSize*, *TowingCapacityPnd*], $T_1$]

| **Model** | **EngineSize** | **TowingCapacityPnd** |
|---|---|---|
| AB99 | _ | _ |

Table 2.3: Sample tableau for variable CFD $\phi_1$

$\phi_2$: [*Model*], [*ManYear*] $\rightarrow$ [*TowingCapacityPnd*], $T_2$]

| **Model** | **ManYear** | **TowingCapacityPnd** |
|---|---|---|
| STON | 2015 | 11300 |
| STON | 2011 | 10250 |

Table 2.4: Sample tableau for constant CFD $\phi_2$

For example, in the data in table 2.1, suppose that there is a constraint that states that if the **Model** for any two truck is "AB99", then they have the same engine size and towing capacity. Table 2.3 shows this variable CFD $\phi_1$ with a tableau $T_1$.

Another example is in table 2.1, in which there is another constraint which states that if the **Make** = "STON" and the **ManYear** = "2015" then the **Towing Capacity** = "13000", and if the **Make** = "STON" and the **ManYear** = "2011" then **TowingCapacityPnd** = "10250". Table 2.4 shows this constant CFD $\phi_2$ with a tableau $T_2$.

### 2.5.3 Inclusion Dependencies

An inclusion dependency (IND) is a type of integrity constraint which can take the form $R_1[X] \subseteq R_2[Y]$ where $R_1$ and $R_2$ are two relations and $X$ and $Y$ are sets of attributes in $R_1$ and $R_2$, respectively, and $X$ and $Y$ have the same number of attributes. An example of inclusion dependency can be specified between the *Truck* relation shown in table 2.1 and the *Orders* relation shown in table 2.5.

*Orders*

| | **OrderNum** | **TruckID** | **DrID** | **DeliveryPlace** | **OrderDate** | **InvoiceNum** |
|---|---|---|---|---|---|---|
| $t_1$ | 72434567 | 11949 | 89 | Toronto | 3 Feb, 2015 | 70122345 |
| $t_2$ | 72434568 | 35246 | 112 | Hamilton | 5 Feb, 2015 | 80126412 |
| $t_3$ | 72434569 | 11949 | 32 | Milton | 5 Feb, 2015 | 80126433 |

Table 2.5: Example of inclusion dependency

*Orders* relation indicates information about orders including the number of order (**OrderNum**), truck ID (**TruckID**), driver ID (**DrID**), delivery place of the order (**DeliveryPlace**), order date (**OrderDate**), and invoice number (**InvoiceNum**). The inclusion dependency is: $Orders[TruckID] \subseteq Truck[Number]$.

### 2.5.4 Conditional Inclusion Dependencies

A conditional inclusion dependency (CIND) $\psi$ can be defined over two relational schemas $R_1$ and $R_2$ as a pair $(R_1[X;\, X_p] \subseteq R_2[Y;\, Y_p],\, T_p])$ where $R_1[X] \subseteq R_2[Y]$ is a standard IND, and $(X,X_p)$ and $(Y,Y_p)$ are disjoint lists of attributes in the attributes of $R_1$ and $R_2$, respectively. $T_p$, referred to as pattern tableau of $\psi$, is a tableau with attributes in $X_p$ and $Y_p$. For each pattern tuple $t_p$ in $T_p$ and for each attribute $B$ in $X_p$ or $Y_p$, the value of $t_p[B]$ is either unnamed variable '_' or constant from the domain of $B$. Inconsistencies could be across different relations in which INDs can be used but in some cases they cannot detect inconsistencies. For example, in figure 2.9, there are three instances of relations: *Shipping*, *Exteriors*, and *Interiors*. The *Shipping* table specifies the shipments of car parts including the shipment number (**Shipment#**), part's name (**PartName**), vehicle brand (**VehicleBrand**), part type (**PartType**), and shipping cost (**ShippCost**). The *Exteriors* table specifies the exterior parts of a car including part ID (**PartID**), the available parts (**PartName**), the brand of vehicle that a part belong to (**Brand**), and the unit price (**UnitPrice**). The *Interiors* table specifies the interior parts of a car including part ID (**PartID**) the available parts (**PartName**), brand of vehicle (**Brand**), and the unit price (**UnitPrice**). If we want to map the *Shipping* relation to *Exteriors* and *Interiors* relations, we might decide to choose the following INDs:

$\text{IND}_1$: $\mathit{Shipping}(\mathit{PartName}, \mathit{VehicleBrand}) \subseteq \mathit{Exteriors}(\mathit{PartName}, \mathit{Brand})$
$\text{IND}_2$: $\mathit{Shipping}(\mathit{PartName}, \mathit{VehicleBrand}) \subseteq \mathit{Interiors}(\mathit{PartName}, \mathit{Brand})$

In this case, a tuple in the *Shipping* is expected to be in the *Exteriors* and *Interiors* tables. This is unrealistic because a tuple in *Exterior* (res. *Interiors*) is not expected to have corresponding *Interiors* (res. *Exterior*) tuples (i.e. pairwise match in the part name and brand attribute values in *Interiors* (res. *Exteriors*)).

*Shipping*

| | **Shipment#** | **PartName** | **VehicleBrand** | **PartType** | **ShippCost** |
|---|---|---|---|---|---|
| $t_1$ | 874353345 | Bumper | FRD Exp | Ext | 90.00 |
| $t_2$ | 456057147 | Seat Belt | TYOT Cmr | Int | 36.00 |
| $t_3$ | 902556732 | Right Front Door | TYOT Prs | Ext | 180.00 |
| $t_4$ | 109285890 | Seat Cover | FRD Exp | Int | 79.0 |
| $t_5$ | 902556732 | Left Front Door | FRD Mus | Ext | 181.0 |

*Exteriors*

| | **PartID** | **PartName** | **Brand** | **UnitPrice** |
|---|---|---|---|---|
| $t_1$ | B9897787 | Bumper | FRD Exp | 120.00 |
| $t_2$ | BD073578 | Right Front Door | TYOT Prs | 155.50 |
| $t_3$ | AZ095120 | Left Front Door | FRD Mus | 165.60 |

*Interiors*

| | **PartID** | **PartName** | **Brand** | **UnitPrice** |
|---|---|---|---|---|
| $t_1$ | SX9127877 | Seat Belt | TYOT Cmr | 40.00 |
| $t_2$ | RX0902122 | Dashboard Cover | TYOT Prs | 110.70 |
| $t_3$ | LK9923909 | Seat Cover | FRD Exp | 80.00 |

Figure 2.9: Example of conditional inclusion dependencies

There should be a condition in the mapping in which the attributes of *Shipping* are mapped to *Exteriors* only if its **PartType** is "Ext" and mapped to *Interiors* only if its **PartType** is “Int”. In this case, the constraints are not inclusion dependencies but conditional inclusion dependencies, as they each have patterns and each have some data values. To capture this, they may be written as:

$\psi_1$: : $\mathit{Shipping}(\mathit{PartName},\ \mathit{VehicleBrand};\ \mathit{PartType}{=}\mathrm{Ext}) \subseteq \mathit{Exteriors}(\mathit{PartName},\ \mathit{Brand}),\ T_1)$
$\psi_2$: : $\mathit{Shipping}(\mathit{PartName},\ \mathit{VehicleBrand};\ \mathit{PartType}{=}\mathrm{Int}) \subseteq \mathit{Exteriors}(\mathit{PartName},\ \mathit{Brand}),\ T_2)$

Tables 2.6 and 2.7, illustrate CIND $\psi_1$, and CIND $\psi_2$, in addition to their tableaux. The standard INDs do not necessarily hold.

$\psi_1$: $\mathit{Shipping}(\mathit{PartName},\ \mathit{VehicleBrand};\ \mathit{PartType}) \subseteq \mathit{Exteriors}(\mathit{PartName},\ \mathit{Brand}),\ T_1)$

| **PartType** | |
|---|---|
| Ext | |

Table 2.6: Sample tableau for CIND $\psi_1$

$\psi_2$: $\mathit{Shipping}(\mathit{PartName},\ \mathit{VehicleBrand};\ \mathit{PartType}) \subseteq \mathit{Interiors}(\mathit{PartType},\ \mathit{Brand}),\ T_2)$

| **PartType** | |
|---|---|
| Int | |

Table 2.7: Sample tableau for CIND $\psi_2$

## 2.6 Data Currency

*Currency* is defined (Redman, 1996) as the degree to which a datum is up-to-date. If the value of a datum is correct despite the possible discrepancies in its values caused by changes related to time, then it is up-to-date. We may use timestamps to determine the current data. For example, consider the instance of a *Nurse* relation in table 2.8. We can identify from the timestamps that $t_2$ is more current than $t_1$ based on the fact that the employee was promoted at a later date in $t_2$ as compared to $t_1$. But in real-life, the time of event occurrences could be imprecise or unknown (Zhang *et al.*, 2010), which would make it harder to identify the current values of attributes.

| | **NurseName** | **Clinic Location** | **PromotionYear** |
|---|---|---|---|
| $t_1$ | Sara Adam | Whitby | 2014 |
| $t_2$ | Sara Adam | Ajax | 2015 |
| $t_3$ | Jack Brown | Burlington | 2012 |
| $t_4$ | Salem Sami | Haliburton | 2011 |

Table 2.8: Timestamps in data

For example, in table 2.9, there are no timestamps, and thus it is unknown whether $t_1$ or $t_2$ is more current. It might be possible to deduce currency orders from the semantics of data (Fan *et al.*, 2011). We may know that a value or a tuple is more current than another. Consider table 2.9 for example. Based on the knowledge that a nurse would have to be promoted to become a lead nurse, we may conclude that the "Lead Nurse" title is more current than the "Nurse" title. Thus, we identify tuple $t_2$ as more current than tuple $t_1$.

| | **NurseName** | **Clinic Location** | **Title** |
|---|---|---|---|
| $t_1$ | Sara Adam | Whitby | Nurse |
| $t_2$ | Sara Adam | Ajax | Lead Nurse |
| $t_3$ | Jack Brown | Burlington | Dental Assistant |
| $t_4$ | Salem Sami | Haliburton | Engineer |

Table 2.9: Data without timestamps

## 2.7 Data Accuracy

*Data accuracy* is the closeness of a value $v'$ in a database to another value $v''$, in which $v''$ is the true value of the entity that the data in the database aims to represent (Fan and Geerts, 2012). For example, table 2.10 represents an instance of a *CD* relation. The value $t_1$[Author] represents an inaccurate value because it does not correspond to any name. By using distance, we may identify that the name "Josephine Harrison" is the closest name to "Josehpine Harriso".

| | **Title** | **Author** | **Genre** |
|---|---|---|---|
| $t_1$ | Oceans | Josehpine Harriso | Stars inc |
| $t_2$ | Walking in the Clouds | Green Sam | Moon publish |
| $t_3$ | Blue Sky | Jack Brown | Ocean inc |

Table 2.10: Inaccurate data

# Chapter 3

# Density-based Deduplication with Data Consistency and Accuracy

In this Chapter, we present our approach for density-based deduplication with data consistency and accuracy. This Chapter is organized as follows: Section 3.1 provides an overview of the approach. In section 3.2, we present our model of weight assignment to the records of the data. In Section 3.3, we describe how to repair the data using our techniques. In section 3.4, we present our technique to incorporate discovery of accurate values into data deduplication process. In section 3.5, we discuss the merging step, followed by section 3.6, in which we discuss the algorithms. Section 3.7 presents related work.

## 3.1 Introduction

A huge amount of data is created and saved everyday in databases from different types of data sources such as financial data, web log data, sensor data, and human input. Different organizations worldwide use data to support their activities through various applications.

The quality of data degrades over time. In reality, data in databases are often dirty. There are main aspects in which issues may cause degradation of data quality. Some of these main aspects include maintaining databases such that the data represent uniquely, consistently, and accurately the entities of the real-world to which they are referring (Fan and Geerts, 2012). To ensure valuable data, issues causing degradation of the data should be resolved. Duplicate records happen when tuples in one or more relations refer to the same real-world entity. The deduplication process has been divided generally in several systems, such as (Galhardas *et al.*, 2001), into three steps: record or tuple matching, clustering of tuples, and merging the tuples of each cluster into a single tuple. Data consistency refers to the integrity and validity of data that represent real-world entities. Integrity constraints such as functional dependencies could be violated for different reasons such as integration with other data sources. This violation makes data inconsistent, i.e. inaccurate. Data accuracy can be defined in general as the closeness of a value $v_1$ in a database to another value $v_2$, in which $v_2$ is a true value of the entity that the data in the database aims to represent (Fan and Geerts, 2012). The following example illustrates how different data quality issues may exist in data. We show cases where we have different issues, and that the benefit of taking all of them into consideration in the cleaning process is crucial to get a better quality of data.

| | **Name** | **JobTitle** | **CreditRank** | **IncomeLevel** | **GeographicLocation** |
|---|---|---|---|---|---|
| $t_1$ | Adam Joe | Sales Director | Silver | A | Windsor ON |
| $t_2$ | Joe Adam | Sales Manager | Silver | A | Ontario |
| $t_3$ | Galvin Vinson | Engineer | Golden | A | Saskatoon SK |
| $t_4$ | Adam Luk Joe | Sales | Golden | A | Windsor ONT |
| $t_5$ | A. Joe | Sales Director | Golden | A | ON Windsor |
| $t_6$ | Jane Mark | Data Analyst | Golden | A | Vancouver BC |
| $t_7$ | Laura Smith | Medical Practitioner | Silver | A | Calgary Alberta |
| $t_8$ | Reema Salem | Chair Man | Silver | A | Brook NL |
| $t_9$ | Maria Erick | Team Leader | Silver | A | Dauphin Manitoba |
| $t_{10}$ | Sara Kim | Marketing Manager | Silver | A | Surrey BC |
| $t_{11}$ | Farah Craig | Nurse | Golden | A | Calgary AB |
| $t_{12}$ | Emery Sawyer | Pilot | Bronze | C | Airdrie AB |

Table 3.1: Instance of *Customers* relation

*Example 3.1*: Consider table 3.1 and the following FD: $\phi_{Customers}$ : *IncomeLevel* $\rightarrow$ *CreditRank*. The *Customers* table contains information about the customers, including the name, job title, credit rank, income level, and geographic location. $\phi_{Customers}$ is a functional dependency, and asserts that income level determines credit rank. We can see that the data in the table are dirty due to data quality issues. The first quality issue is that there may be duplicate records in the table that may refer to the same entity or customer. The tuples $t_1$, $t_2$, $t_4$ and $t_5$ could refer to the same customer. We need to make a decision about the correct values that represent the entity, that is, a final single tuple. The second quality issue is that the tuples $t_1$, $t_2$, $t_4$ and $t_5$ violate $\phi$ which makes the data inconsistent regarding these tuples. We need to repair this inconsistency by either changing $t_1$[CreditRank], and $t_2$[CreditRank] to "Golden" or changing $t_4$[CreditRank] and $t_5$[CreditRank] to "Silver", but there is not clear evidence to tell us which change to choose. Some of these tuples might have more correct values than others; that is, there is more confidence in their values, but it is expensive for the user to assign the confidence scores manually. The third quality issue is that it is not clear which value in **GeographicLocation** is the most accurate in the tuples $t_1$, $t_2$, $t_4$ and $t_5$. We need to find out whether "Windsor ON", "Ontario", "Windsor ONT", or "ON Windsor" is the most accurate location in the final single tuple. Similarly, we need to decide on the most accurate name and job title in **Name** and **JobTitle**, respectively. Decisions about confidence of correctness of the tuples in the data would help in repairing the inconsistency in **CreditRank** and **IncomeLevel**, which would help then in the decision about the final single tuple. Moreover, discovering the accurate values **GeographicLocation**, **Name**, and **JobTitle** would also help in this decision. Thus, the deduplication process would become more efficient by solving the above issues all together which, in turn, would lead to better data cleaning.

Previous supervised and active learning-based techniques in duplicate elimination (Cohen and Richman, 2002; McCallum and Wellner, 2004; Sarawagi and Bhamidipaty, 2002) required either training data or manual user interaction. Other techniques in duplicate elimination such as distance-based techniques (Hernández and Stolfo, 1998) and unsupervised learning techniques (Ravikumar and Cohen, 2004) tried to solve the drawbacks of the supervised and active-learning-based techniques. However, they do not repair integrity constraints violations in the data. Work on repairing integrity constraints violations has been reported previously (Bohannon *et al.*, 2005; Kolahi and Lakshmanan, 2009). These techniques are based on the minimum change principle that uses cost-based models to find a minimum change. In (Bohannon *et al.*, 2005), an approach that is based on the minimal change was proposed to clean the data. It connects record linkage with integrity constraints repair. The integrity constraints (FDs) are repaired through value modification based on a similarity distance metric of record linkage techniques. The accuracy of the data in this approach is reflected as weights of tuples that are set manually by the user. The work of (Kolahi and Lakshmanan, 2009) follows a similar weight model. Another data repair approach that deals with consistency and accuracy which is also based on minimal change to repair CFDs is proposed by (Cong *et al.*, 2007). The user is also involved in this technique in the assignment of weights to the attribute values. The techniques in (Bohannon *et al.*, 2005; Cong *et al.*, 2007; Kolahi and Lakshmanan, 2009) do not consider local structural properties in the assignment of weights while cleaning the data. Other work by (Fan *et al.*, 2014b) for data cleaning is based on constraint theory. It is based mainly on the availability of confidence scores assigned by the user and the availability of master data. Work on data accuracy and truth discovery from data has been reported previously (Cao *et al.*, 2013; Dong *et al.*, 2009; Wu and Marian, 2011; Yin *et al.*, 2008). Other works perform word clustering (Dhillon *et al.*, 2002; Slonim and Tishby, 2001). However, these techniques are either based mainly on manual user interaction and a master data (a.k.a. reference data) (Cao *et al.*, 2013), or they do not consider the duplicate elimination problem (Dhillon *et al.*, 2002; Dong *et al.*, 2009; Slonim and Tishby, 2001; Wu and Marian, 2011; Yin *et al.*, 2008).

Although consulting the user to help in decisions about quality issues or relying on a master data are possible solutions, these approaches have their limitations. Master data is usually unavailable in practical situations (Batini and Scannapieca, 2006), and proper training data is hard to obtain Elmagarmid *et al.* (2007). In addition, the manual process of editing data such as assigning weights by users is time consuming and unrealistic in practice, as databases are usually large in size. As a matter of fact, it may take months to manually clean census data by dozens of clerks (Winkler, 2004). In this work, we argue that in order to make decisions about the correctness of the attribute values in data, the deduplication approach may use the density of data to differentiate between tuples that have correct values versus erroneous values. We present a new framework of data cleaning that improves the quality of data by taking into consideration three aspects of data quality, namely, data deduplication, data consistency, and data accuracy. Thus, data cleaning is achieved without a need for human effort, master data, or training data.

Returning to example 3.1, a density-based clustering algorithm may compute the density of data in order to assign confidence scores. The density associated with a tuple is obtained by taking into consideration the number of tuples in a region within a specified radius around the tuple, which is detailed in section 3.2. In table 3.1, the tuples, i.e. data points, $t_1$, $t_2$, $t_4$, and $t_5$ may belong to the same cluster. However, tuples $t_1$, $t_2$, and $t_5$ may lie in a higher density region so there is more confidence, i.e. higher weights, in the correctness of their values than tuple $t_2$. That is, we may infer the correct attribute values of the tuples based on data density. For example, the value of $t_5$[CreditRank] may have higher confidence score than the $t_2$[CreditRank] value.

This work addresses the problem of data cleaning for duplicate records in a database. These records have inconsistent values due to violations in functional dependencies and inaccurate values resulting from transcription errors and a lack of standard formats. This functionality integrates data repair with data accuracy in a density-based deduplication. We utilize the density-based information that are already embedded in the data to assign weights for tuples, and then use the assigned weights to compute the cost of data repairs. We also use the clustering results in the discovery of the accurate values. Finally, we merge the resultant values into a single tuple.

We focus on functional dependencies (FDs) in this work because they are a widely used form of integrity constraints, which in practice, are commonly violated. We also assume a model with NULL marks with a closed world assumption (CWA) where some attribute values of a tuple could be missing (NULL). In addition, no manual user interaction is involved, and we assume the availability of neither master data nor training data. We make the following contributions:

- A weight model that is based on the density of data to determine the weights of the tuples. The weights represent the confidence in the correctness in the values of the data.
- A cost model that is based on the density of data for repairing a database that is inconsistent with respect to a given set of integrity constraints.
- A uniform data cleaning framework and algorithm that integrate data deduplication with inconsistent data repairing and discovering of the accurate values in the data. The fixes of data quality issues are based on the density of data.
- We evaluate experimentally the data cleaning framework and algorithms using synthetic and real datasets. We do experiments to check the quality of data cleaning in terms of precision (correctness of cleaning) and recall (completeness of cleaning). In addition, we evaluate the scalability using various experiments. We show that our framework and algorithms can effectively clean data.

## 3.2 Density-based Weight Model

We cluster the objects that are the records in our model in a database $D$ of points of some data space $S$, using DBSCAN (Ester *et al.*, 1996), a density-based clustering algorithm. This algorithm has other strengths that make it effective in deduplication. It is robust concerning outliers, it discovers arbitrary shaped clusters, and it is efficient (Januzaj *et al.*, 2003). Moreover, it does not require specification of the number of clusters *a priori*. Determining the number of clusters in advance could degrade the quality of a clustering algorithm in detecting duplicates, as the number of duplicate entities is usually unknown in the data. Given a set of objects (tuples) $I$, the clusters are considered as regions in some data space $S$ in which a given set of objects $O$ are dense. These dense objects represent the duplicate records or tuples in our model. The objects that lie in low density regions are outliers. They separate dense objects and represent the records that are not duplicates in our model. In the algorithm, there is a given neighborhood, denoted by $\varepsilon$, which is a radius within which a minimum number of objects, denoted by $MinPts$, have to be contained. In other words, the neighborhood's cardinality is $\geq$ some threshold. We use the following definitions (adapted from (Ester *et al.*, 1996; Januzaj *et al.*, 2003)) to explain DBSCAN.

An object $x$ is *directly-reachable* from an object $y$ w.r.t. $\varepsilon$ and $MinPts$ in the set of objects $O$, if $x \in$ neighborhood of $y$ and cardinality of the neighborhood of $y \geq MinPts$. An object $x$ is *density-reachable* from an object $y$ w.r.t. $\varepsilon$ and $MinPts$ in the set of objects $O$, if there is a chain of objects $x_1, x_2, ..., x_n$, $x_1 = y$, $x_n = x$, such that $x_i \in O$ and $x_{i+1}$ is directly-reachable from $x_i$ w.r.t. $\varepsilon$ and $MinPts$. An object $x$ is *density-connected* from an object $y$ w.r.t. $\varepsilon$ and $MinPts$ in the set of objects $O$, if both $x$ and $y$ are density reachable from an object $o \in O$ w.r.t. $\varepsilon$ and $MinPts$ in $O$.

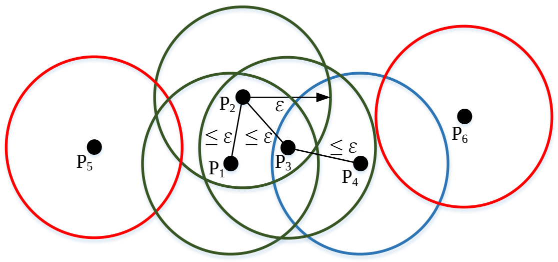

Figure 3.1: Core, border, and outliers tuples

A cluster $C$ is defined as a non-empty subset of $O$ that satisfies the following two conditions: 1) $\forall x, y \in O$: if $x \in C$ and $y$ is density reachable from $x$ w.r.t. $\varepsilon$ and $MinPts$, then $y \in C$. 2) $\forall x, y \in C$: $x$ is density connected to $y$ w.r.t. $\varepsilon$ and $MinPts$. When a tuple is inside of a cluster where the number of neighbors is at least $MinPts$ within $\varepsilon$, it is called a *core tuple*. Otherwise, the tuple is called a *non-core tuple*. The non-core tuples are divided into either *border tuples* or *outliers*. Border tuples are part of the cluster and a tuple is a border tuple when it is density reachable from another core tuple. A tuple is an *outlier* when it is neither a core tuple nor reachable from another core tuple.

Figure 3.1 illustrates these definitions. With $MinPts = 3$ and distance $\varepsilon$, a cluster $C$ is formed with four tuples $p_1$-$p_4$. Tuples $p_1$-$p_3$ are core tuples as there are at least three tuples within $\varepsilon$ for each of them, that is, there is high density region within $\varepsilon$. Tuples $p_4$ is a border tuple, while tuples $p_5$ and $p_6$ are outliers.

In a density-based algorithm, the data space is divided into regions where there are different densities of objects. We use this fact to assign a weight for each tuple. Assigning weights using the density of data would leverage the confidence about which of the attribute values may have the correct values. This is based on the intuition that the density in the region of the core tuples is higher than the density of the region of the border tuples so that there is more confidence in the assumption that the region with the higher density would have the correct values. For example, in figure 3.1, the density of the region of tuple $p_1$ is higher than that of tuple $p_2$, so there is more confidence in the attribute values of tuple $p_1$. We also use this fact in the cost model introduced in section 3.3.

For a relational schema $R$, let $T$ denote a set of $n$ attributes $\{A_1, A_2, ..., A_n\}$ of $R$. The domain of an attribute $A \in R$ is denoted by $dom_A$. A set of tuples that is an instance of $R$ is denoted by $I$, in which each tuple is in the domain $dom_{A_1} \times ... \times dom_{A_n}$. Let $N$ be the number of tuples (cardinality) of $I$ or $|I|$. We denote the value of an attribute $A \in R$ of a tuple $t \in I$ by $t[A]$. We may also refer to $t[A]$ as *cell*. Let $Q$ be a set of $q$ clusters $\{C_1, C_2, ..., C_q\}$ that are clustered according to the density-based clustering algorithm where each cluster is a set of distinct tuples in $I$. Let $I_c$ be an instance of the tuples of a singleton set $E$ of cluster $C \in Q$. Let $I'_c$ be an instance of all tuples of the remaining records in $I$ where $I'_c = I$ - $I_c$. We build on previous work in data repair techniques (Bohannon *et al.*, 2005) and assume that there is a weight $w(t) > 0$ for each tuple $t \in I_c$ that is associated with $t$. The weight reflects the confidence in the correctness of the values of the tuple. Let $k$ and $b$ be the numbers of the core and border tuples, respectively, in $I_c$. We define the weight of a core tuple $t_k \in I_c$ and the weight of a border tuple $t_b \in I_c$, where a higher score indicates more confidence in the correctness of the values of a tuple, as:

**Definition 1.** *Core Tuple Weight ($w(t_k)$) is the number of core tuples divided by total number of tuples in $I_c$.*

$$w(t_k) = \frac{k}{|I_c|}$$

**Definition 2.** *Border Tuple Weight ($w(t_b)$) is the number of border tuples divided by* *total number of tuples in $I_c$.*

$$w(t_b) = \frac{b}{|I_c|}$$

*Example 3.2*: In table 3.1, let's assume that tuples are objects in some data space where $MinPts = 3$ and $\varepsilon = 0.6$. The algorithm may cluster tuples $t_1$, $t_4$, and $t_5$ as core tuples, so they form a single cluster $C$, and tuple $t_2$ as a border tuple so it also belongs to $C$. Tuples $t_3$ and $t_6$-$t_{12}$ are outliers and they do not belong to cluster $C$. The weights of tuples $t_1$, $t_2$, $t_4$, and $t_5$ are 0.75, 0.25, 0.75, and 0.75, respectively.

## 3.3 Data Repair

A functional dependency (FD) $F : X \rightarrow Y$ defined over a relation $R$ means that the values of a set of attributes $X$ in a tuple uniquely or functionally determine the values of another set of attributes $Y$ of the tuple. That is, for any two tuples $t'$ and $t''$, if $t'[X] = t''[X]$ then $t'[Y] = t''[Y]$. For a set of functional dependencies $\Sigma$, $I \models \Sigma$ ($I$ satisfies $\Sigma$) iff none of the tuples in $I$ violate any F in $\Sigma$. This means that, if $t'[X]$ and $t''[X]$ are in a class $[x]$, that is, sharing same $X$ values, then $t'[Y]$ and $t''[Y]$ should be in the same class $[xy]$, that is, sharing the same $Y$ values. Class $[xy]$ is a refinement of $[x]$. We assume that each functional dependency $F : X \rightarrow Y$ contains a single attribute in $Y$, that is, $\Sigma$ is minimal (Abiteboul *et al.*, 1995). We also assume that the attribute sets of the FDs are disjoint sets. Let set $Z$ be a union of $X$ and $Y$.

### 3.3.1 Cost Model of Data Repair

We use a cost model to assess the cost of repair and how to modify different values to repair inconsistency. We find the lowest cost changes, i.e., the minimal repairs, for an inconsistent database (Bohannon *et al.*, 2005; Cong *et al.*, 2007; Kolahi and Lakshmanan, 2009). We assume that we have an FD $F : X \rightarrow Y$. We first seek for the violations of FDs in each cluster in the case $t'[Y] \neq t''[Y]$. That is, for one class $[x]$, there is more than one class $[xy]$ where each of $[xy]$ classes shares the same $X$ values but differs in $Y$ value. We repair such violations by choosing one of the $[xy]$ classes and changing the values of $Y$ of other $[xy]$ classes to the $Y$ value of the chosen class, depending on the repair cost. We end up with one $Y$ value for each $[x]$ class. After repairing $[xy]$ classes, as our goal is to get one tuple that represents the entity, we seek how to repair $[x]$ classes in the case where there is more than one class $[x]$ in the cluster. We let $p_{xy}$ be a representative of a class $[xy]$, that is, $p_{xy}$ is an element of $[xy]$. Let $p_x$ be a representative of a class $[x]$. Our cost approach is based on repairing inconsistent attribute values of each tuple in $I_c$ in each $C \in Q$. We consider three factors in this approach to measure the cost. The first factor is the weight of each representative. The second factor is the frequency of each representative of class $[xy]$ (resp. $[x]$) $\in I_c$ which occurs in $I'_c$. The third factor is the distance measure between the attribute values of the representatives in each class $[xy]$ (resp. $[x]$). The weight of a representative $p_{xy}$ is defined as:

**Definition 3.** *Weight of representative ($w(p_{xy})$) is the summation of weights of the core tuples plus the summation of the weights of the border tuples where the representative exists.*

$$w(p_{xy}) = \sum_{i=1}^{m_c} w(t_k) + \sum_{k=1}^{m_b} w(t_b)$$

where $m_c$ and $m_b$ are the numbers of times that $p_{xy}$ exists in the core and border tuples of a cluster, respectively. While our data repair algorithm is similar in spirit to the techniques in (Bohannon *et al.*, 2005; Cong *et al.*, 2007; Kolahi and Lakshmanan, 2009), our density-based weight technique has added benefits in assigning weight without the need for manual user input. To decide on the frequency of a representative of class $[xy]$, i.e. the second factor, let $L = \{p_{xy_1}, ..., p_{xy_m}\}$ be a set of representatives of each class $[xy] \in I_c$. Let $f_{p_{xy}}$ be the number of times that each representative $p_{xy}$ in $L$ occurs in $I'_c$. Let $f'_{p_{xy}}$ be the number of times that all the representatives in $L$ occur in $I'_c$. We define the frequency of how many times each representative $p_{xy}$ in $L$ occurs in $I'_c$, where a higher value indicates more support in the data, as:

**Definition 4.** *Frequency of representative ($freq(p_{xy})$) is $f_{p_{xy}}$ divided by $f'_{p_{xy}}$.*

$$freq(p_{xy}) = \frac{f_{p_{xy}}}{f'_{p_{xy}}}$$

The choice of distance measure $d$, i.e. the third factor, between the $Y$ values of the representatives may depend on the domain of the data type of the attribute values. Hamming distance could be used for numerical fixed size strings of attribute values (Batini and Scannapieca, 2006). For variable length of strings of attribute values, Jaro-Winkler (Winkler, 1999) distance may be used. In Jaro-Winkler distance, the distance between two values of representatives, $p'_{xy}$ and $p''_{xy}$ is defined such that a higher score indicates high similarity and vice versa. A score of "1" indicates an exact match or $p'_{xy} = p''_{xy}$ while a score of "0" indicates no similarity. We measure the overall distance measure to change every $Y$ value in the class of each representative into the other values of the other representatives in order to compare potential changes. We multiply the distance measure by the number of times the representative that we want to change exists in the cluster. We denote this number by $j_m$. The class that has the lowest cost to change the values of other classes to its value will be the chosen class. That is, our algorithm is greedy in which it selects the lowest cost for repairs.

We end up with the best, i.e., chosen, $[xy]$ class for each cluster. Let $H$ be a singleton set of a representative $p_{xy}$. Let $P = L$ - $H$ and let $f = |P|$. We define the cost as:

**Definition 5.** *Cost to change to class $[xy]$ is the cost to change the $Y$ value of each class $[xy']$ of representative $p'_{xy} \in P$ to the $Y$ value of class $[xy]$.*

$$Cost([xy]) = \sum_{i=1}^{f}(w(p'_{xy}) * (\sum_{k=1}^{j_m} d(p_{xy}, p'_{xy})) * freq(p'_{xy}))$$

*Example 3.3*: Consider table 3.1, let the tuples be clustered as in example 3.2 and let FD $\phi_{Customers}$ : IncomeLevel $\rightarrow$ CreditRank. The tuples in the cluster violate $\phi$. $I_c = \{t_1, t_2, t_4, t_5\}$ and $I'_c = \{t_3, t_6, t_7, t_8, t_9, t_{10}, t_{11}, t_{12}\}$. In $I_c$, we have one class $[x]$ where the value of X =“A” and two classes $[xy_1]$ and $[xy_2]$, one with X =“A” and Y = “Golden’ and another with X =“A” and Y =“Silver”, respectively. So we have two representatives, $p_{xy_1}$ and $p_{xy_2}$ for each $[xy]$ class, respectively. We use Jaro-Winkler distance in this example.

- There are three core tuples and one border tuple. The weight of each core tuple = 3/4 = 0.75. The weight of border tuple = 1/4 = 0.25
- The weight of $p_{xy_1}$ = 2 * 0.75 = 1.5. The weight of $p_{xy_2}$ = 1 * 0.75 + 1 * 0.25 = 1
- We measure the distance between the two representatives. To change $p_{xy_2}$ to $p_{xy_1}$, $d$(‘Golden’,‘Silver’) = 0.555. We have $j_m = 2$ so $d$ multiplied by $j_m$ = 2 * 0.555 = 1.111. To change $p_{xy_1}$ to $p_{xy_2}$, $d$(‘Silver’,‘Golden’) = 0.555. We have $j_m = 2$ so $d$ multiplied by $j_m$ = 2 * 0.555 = 1.111
- $freq(p_{xy_1})$ = 3/7 = 0.428. $freq(p_{xy_2})$ = 4/7 = 0.571

- Cost($[xy_1]$) = 1 * 1.111 * 0.571 = 0.634. Cost($[xy_2]$) = 1.5 * 1.111 * 0.428 = 0.713
- Since Cost($[xy_2]$) > Cost($[xy_1]$), we choose to change $[xy_2]$ to $[xy_1]$

Now, we have ended up with one $Y$ value for each class $[x]$. In the case where there is more than one class $[x]$ in a cluster, we need to decide which one of them is the final one that has the correct values for $X$ and $Y$. We follow a similar cost model for $[x]$ classes. The weight of class $[x]$ is the weight of its chosen class $[xy]$. The frequency of class $[x]$ is the summation of the frequencies of its $[xy]$ classes. Using a similar distance measurement concept, we measure the distance but now between both $X$ and $Y$ values of $[x]$ classes. Then we compute the cost to change to different classes in the same way as in definition 5. However, this time between $[x]$ classes using their representatives, weights, similarity measures multiplied by the number of times that each class $[x]$ exists in the cluster, and their frequencies.

## 3.4 Discovery of Accurate Values

We have clustered the tuples based on their density. However, reasons other than integrity constraints violations may cause duplicates in which there is a need to discover the accurate values of the attributes in the duplicate tuples. Errors are introduced in duplicate records as the result of transcription errors, lack of standard formats, incomplete data, or any combination of these factors (Elmagarmid *et al.*, 2007). Other reasons, such as different times of data entry or different requirements for data entry may also cause duplicates. For example, the address of a customer may change over time (Naumann and Herschel, 2010). We now discover the accurate values in the attributes that have types of errors such as transcription errors or lack of standard formats without integrity constraints defined over these attributes.

We discover which values are more accurate in a set of attributes $\Gamma$ in a cluster $C$ by measuring the relatedness of the words of the attributes in the tuples that belong to $C$. Each attribute value of a tuple is a string that is composed of a set of tokens (or words). For example, in table 3.1, $t_1$[GeographicLocation] consists of two tokens: "Windsor" and "ON". Our objective is to discover the most accurate value that represents an attribute in a cluster $C \in Q$; we call this the *Accuracy Attribute* and denote it by $A_c$. Let $U$ be a set of accuracy attributes where $U$ and $Z$ are disjoint sets. We employ a bag-of-words model to measure the relatedness of the attribute values (i.e. observations). The occurrence of each token or feature is counted and noted in a matrix where the rows represent each token and the columns represent the occurrences of the tokens in each tuple of the cluster. We use feature selection as it is better than word clustering at removing detrimental, noisy features (Baker and McCallum, 1998). We preserve the order of 2-grams of words in addition to individual words or 1-grams. For example, in the attribute **GeographicLocation**, if we have the following values in three different tuples: "BC Surrey", "Surrey BC", and "Surrey BC", then the last two would be clustered before the first one. After applying the bag-of-words model on the values, we get the matrix $M$ of features. Table 3.2 illustrates the matrix $M$ for the accuracy attribute **GeographicLocation** of table 3.1.

| **Token\Tuple** | $t_1$ | $t_2$ | $t_4$ | $t_5$ |
|---|---|---|---|---|
| ONT | 0 | 0 | 1 | 0 |
| Windsor | 1 | 0 | 1 | 1 |
| Ontario | 0 | 1 | 0 | 0 |
| ON | 1 | 0 | 0 | 1 |
| Windsor ON | 1 | 0 | 0 | 0 |
| Windsor ONT | 0 | 0 | 1 | 0 |
| ON Windsor | 0 | 0 | 0 | 1 |

Table 3.2: Matrix $M$ of occurrences of tokens over tuples

The presence of a value in a tuple is indicated by the number of its occurrences and the absence of any occurrence of a value is indicated by 0. We then apply a greedy agglomerative clustering (Kaufman and Rousseeuw, 2005) in which each observation starts in its own cluster, then clusters that co-occur are merged. These clusters that co-occur represent the most accurate values of an attribute. The agglomerative clustering starts with a collection $\mathbf{G}$ of $g$ singleton clusters where each singleton cluster has one observation. Then at each step, it combines or merges pairs of clusters that are the closest according to a distance metric *dist*. To decide which clusters to combine, we specify a linkage criterion and a metric that is used to compute the linkage. The choice of the metric depends on the domain. For example, a token-based distance metric could be used in order to measure the distance between pairs of tokens. Token-based distance metrics have the ability to handle the rearrangement of words (Elmagarmid *et al.*, 2007), that is, it is not sensitive to word swaps. An example of such metrics is classical Jaccard distance (Jaccard, 1901) (which is a special case of Marczewski-Steinhaus distance distance[1] (Marczewski and Steinhaus, 1958)). Jaccard distance compares strings and we use it to find how two tokens are dissimilar; that is, the larger the distance between tokens, the less likely that they are close to each other. For two strings $A$ and $B$, Jaccard distance *dist*, is given by:

[1]It has recently been shown (Janicki and Lenarcic, 2016) that Marczewski-Steinhaus distance represents a class of equivalent distances, however, these results have not been used in this thesis.

$$dist(A, B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}$$

For the linkage criterion or merging approach, we use an average-link linkage because it performs well in practice, the data are not restricted to Euclidean distance, and it is reasonably robust (Kaufman and Rousseeuw, 2005). In the average-link linkage, the distance between two pairs of clusters is the average of all pairwise distances between the tokens of each cluster. The pairs of clusters that minimize this criterion will be merged. The distance $D$ for two clusters $C_1$ and $C_2$, is given by:

$$D(C_1, C_2) = \frac{1}{|C_1|.|C_2|} \sum_{x \in C1} \sum_{y \in C2} dist(x, y)$$

The resultant clusters of the clustering are ordered in a hierarchical tree called a dendrogram as the merge moves up the hierarchy. The clusters that are closest to each other will be clustered before the clusters that are less similar to each other. We place a threshold value $\theta$ to dictate how much similarity is required to get an accurate value. We then merge all the tokens in the resultant cluster, and then eliminate the duplicate values. The issue of high dimensionality that is usually a characteristic of text data is not substantial in this technique, as we perform the agglomerative clustering on each $C$. This lowers the number of features per cluster, resulting in reduced dimensionality.

It is notable that this discovery of accurate values technique which uses the results of the density-based clustering is complementary to our deduplication technique using data repair. The deduplication can just use the data repair technique, or it can use both of them. However, it was developed to be integrated with the data repairing for deduplication in a density-based data cleaning as a newly introduced functionality. In terms of the problem of discovery of accurate values, the clusters that are closest would be more similar and thus are more likely to be the accurate values. We define the accurate value of an attribute $A_c$ as:

**Definition 6.** *Accurate Value of $A_c$, denoted by $A_{cv}$, is the set of tokens that represents the largest set of clusters that have been clustered within a specified threshold value $\theta$.*

*Example 3.4*: Consider table 3.1 where the tuples are clustered as in example 3.2. Let $\theta = 0.5$. We need to discover each $A_{cv}$ for the accuracy attributes **GeographicLocation**, **Name**, and **JobTitle**.

- In the attribute **GeographicLocation**, the values "Windsor ON" and "ON Windsor" will be merged within a distance not greater than $\theta$. We then merge the tokens of the resultant cluster (i.e. "Windsor ON" and "ON Windsor"), and then eliminate the duplicate values. We find that the most accurate value is "Windsor ON".
- For **Name** and **JobTitle** attributes, the accurate values are "Adam Joe" and "Sales Director", respectively.

## 3.5 Tuple Coalescing

The goal of the deduplication process is to identify a single tuple that has the correct values. That is, for a cluster $C$ of tuples, the goal is to merge the tuples into a single tuple that has the correct attribute values. Merging tuples is usually domain specific in which domain specific rules are applied to merge the tuples. In our approach, choosing the right distance measures according to the domains of the data would improve the cleaning. The results of data repairing and discovery of accurate values are ready to be merged. We define the merged tuple as:

**Definition 7.** *Merged Tuple is the tuple that has the final chosen class $[x]$ in $C$ for each FD and the most accurate value $A_{cv}$, of each $A_c$ in $C$, for each $C \in Q$.*

For example, assuming that the tuples of table 3.1 are clustered as in example 3.2, the *Merged Tuple* after repairing the values of the attributes and discovering the accurate values in $C$ would be (“Adam Joe”,“Sales Director”,“Golden”,“A”,“Windsor ON”).

## 3.6 Data Cleaning Algorithm

In algorithm 1, namely COD (**C**leaning based **O**n **D**ensity), the cleaning of the dirty data $D_d$ starts by running the density-based clustering procedure w.r.t. $MinPts$ and $\varepsilon$ (line 2) which returns a set $Q$ of clusters, a set $O$ of core tuples, a set $B$ of border tuples, and a set $N$ of the outliers (not duplicate) tuples. Then, for each cluster, we compute the weight of core and border tuples (line 4) and then perform data repair and discovery of accurate values (lines 5-6). After that, the tuple coalescing or merging procedure starts where the cleaned attribute values of the tuples for each cluster are returned in the list $D_{coalesce}$ (line 7). Then $D_{coalesce}$ is inserted in $D_c$ (line 8). After finishing with the clusters, $N$ is inserted in $D_c$ to get the final clean data $D_c$ (line 9).

**Algorithm 1** COD

```
Input:  dirty data $D_d$
        set of functional dependencies $\Sigma$
        set of accuracy attributes $\Gamma$
        accuracy threshold $\theta$
        minimum number of neighborhood $MinPts$
        neighborhood radius $\varepsilon$
Output: clean data $D_c$
 1: $D_{coalesce}$, $D_{cons}$, $D_{acc}$ = [ ], [ ], [ ]
 2: $Q$, $O$, $B$, $N$ = clusterTuples($D_d$, $MinPts$, $\varepsilon$)
 3: for each cluster $C \in Q$ do
 4:   $W$ = weightTuples($C$, $O$, $B$)
 5:   $D_{cons}$ = repairData($D_d$, $\Sigma$, $Q$, $C$, $W$, $O$, $B$, $N$)
 6:   $D_{acc}$ = discoverAccurateData($\theta$, $\Gamma$,$C$)
 7:   $D_{coalesce}$ = tuplesCoalesce($D_{cons}$, $D_{acc}$)
 8:   insert $D_{coalesce}$ into $D_c$
 9: insert $N$ into $D_c$
10: return $D_c$
```

Algorithm 2 computes the weights of core and border tuples for each cluster according to the method in section 3.2. It finds the number of core and border tuples (lines 2-3), then it computes the weight for each tuple (lines 5-8). In line 9, it assigns the weight for the tuple in the list $W$.

**Algorithm 2** weightTuples($C$, $O$, $B$)

```
 1: $W$ = [ ]
 2: $k$ = findCoresNumber($O$, $C$)
 3: $b$ = findBordersNumber($B$, $C$)
 4: for each tuple $t \in C$ do
 5:   if $t \in O$ then
 6:     $w = k/|C|$
 7:   else
 8:     $w = b/|C|$
 9:   $W$ = markTupleWeight($w$, $t$)
10: return $W$
```

**Algorithm 3** repairData($D_d$, $\Sigma$, $Q$, $C$, $W$, $O$, $B$, $N$)

```
 1: $D_{cons}$, $rL$, $costXY$, $classXY$
    $costX$, $bestClassX$ = [ ], [ ], [ ], [ ], [ ], [ ]
 2: for each $F \in \Sigma$ do
 3:   $rL$ = getRepr($C$, $F$, $O$, $B$)
 4:   for each $p_x$ in $rL$ do
 5:     for each $p_{xy}$ in $p_x$ do
 6:       $costXY$.append(cost($D_d$, $rL$, $p_{xy}$, $C$, $W$, $O$, $B$, $N$))
 7:     $classXY$.append(min($costXY$))
 8:   for each $p_x$ in $classXY$ do
 9:     $costX$.append(cost($D_d$, $rL$, $p_x$, $C$, $W$, $O$, $B$, $N$))
10:   $bestClassX$ = min($costX$)
11:   insert $bestClassX$ into $D_{cons}$
12: return $D_{cons}$
```

cost($D_d$, $rL$, $p$, $C$, $W$, $O$, $B$, $N$)

```
1: $w_p$ = computeReprWeight($W$, $C$, $O$, $B$, $rL$, $p$)
2: $freq_p$ = computeReprFreq($D_d$, $C$, $N$, $rL$, $p$)
3: $d$ = computeDistance($C$, $rL$, $p$)
4: $cost_p$ = computeChangeCost($w_p$, $d$, $freq_p$, $C$, $rL$, $p$)
5: return $cost_p$
```

Algorithm 3 repairs the inconsistent values, and returns a list of consistent values $D_{cons}$. For each FD, a list $rL$, saves the representatives of classes $[x]$ and $[xy]$ (line 3). Then in lines 4-7, for each class $[x]$, the algorithm first computes the costs of its $[xy]$ classes by calculating the cost to change their values to those of different classes. After that, the class with the best values into which to change the values of other classes is chosen according to the minimum cost among $[xy]$ classes. In lines 8-10, the best class among $[x]$ classes is chosen so we have the final repaired value for the FD which is then inserted in the list $D_{cons}$ (line 11).

**Algorithm 4** discoverAccurateData($\theta$, $\Gamma$, $C$)

```
1: $D_{acc}$, $A_{cv}$ = [ ], [ ]
2: for each $A_c$ in $\Gamma$ do
3:   $M$ = buildMatrixM($A_c$, $C$)
4:   $A_{cv}$.append(clusterAgg($M$, $A_c$, $\theta$))
5:   eliminateDuplicateValues($A_{cv}$)
6: insert $A_{cv}$ into $D_{acc}$
7: return $D_{acc}$
```

The procedure cost in algorithm 3 returns the cost to change to a class. The weight of the representative, the distance, its frequency, and its cost are computed (lines 1-4).

Algorithm 4 returns in the list $D_{acc}$ the accurate values of the accuracy attributes. For each accuracy attribute, the matrix $M$ is computed (line 3), then the agglomerative clustering starts where the most accurate value $A_{cv}$ is discovered (line 4). The duplicate accurate values are eliminated in line 5. In line 6, $A_{cv}$ values are inserted in the list $D_{acc}$.

## 3.7 Related Work

Work on duplicate detection that is related to our research has been reported previously (refer to (Elmagarmid *et al.*, 2007; Winkler, 2006) for surveys). Supervised learning techniques (Cohen and Richman, 2002; McCallum and Wellner, 2004) have the issue of requiring training datasets of known duplicates, prelabeled as matches or not, that have natural correlations among the values. Active learning-based techniques (Sarawagi and Bhamidipaty, 2002; Tejada *et al.*, 2002) try to address this issue, but they have the drawback of requiring human input. Distance-based duplicate detection techniques (Hernández and Stolfo, 1998; Monge and Elkan, 1997) use distance metric and a global matching threshold without the need for a training dataset. Another distance-based technique by (Chaudhuri *et al.*, 2005) proposes an algorithm for duplicate detection that computes distance thresholds for tuples of the same real-world entity that have different representations in the database instead of an absolute global threshold. Unsupervised learning techniques (Ravikumar and Cohen, 2004; Song *et al.*, 2015; Verykios *et al.*, 2000) use clustering algorithms to avoid manual labeling. However, the techniques (Chaudhuri *et al.*, 2005; Hernández and Stolfo, 1998; Monge and Elkan, 1997; Ravikumar and Cohen, 2004; Verykios *et al.*, 2000) do not repair the issues in integrity constraints violations.

Systems have been developed for duplicate detection (Galhardas *et al.*, 2001; Raman and Hellerstein, 2001). The user is usually involved in specifying which actions are required. In (Galhardas *et al.*, 2001), a declarative language is proposed to remove duplicates during data transformations. The logical and physical levels in their framework are separated. In the logical level, the user specifies the logical workflow while in the physical level, the system selects the best implementation in terms of performance. In (Raman and Hellerstein, 2001), the data cleaning system integrates the transformation of data and discrepancy detection steps. Users build transformation interactively through a graphical user interface. A work by Song *et al.* (2015) performs density-based clustering and repairing over unclean data simultaneously. However, it is developed to improve clustering rather than repairing, and it does not merge tuples.

Repairing approaches for data cleaning using integrity constraints have been studied by others: FDs (Beskales *et al.*, 2010; Kolahi and Lakshmanan, 2009), conditional functional dependencies (CFDs) (Cong *et al.*, 2007; Yakout *et al.*, 2011), FDs, CFD, ETL and Business Rules (Dallachiesa *et al.*, 2013), editing rules (Fan *et al.*, 2012), FDs and inclusion dependencies (INDs) (Bohannon *et al.*, 2005), CFDs and matching dependencies (MDs) (Fan *et al.*, 2014b). In (Bohannon *et al.*, 2005; Kolahi and Lakshmanan, 2009), the weights are assumed to be assigned manually to the data by the user. In (Cong *et al.*, 2007), a sampling process is proposed to improve the repair approach where a user manually updates the sample data or inputs new CFD. They also assume a weight that is assigned by the user to the data. In (Beskales *et al.*, 2010), a confidence model is suggested that allows the user to specify which cell should not be changed or to specify the order of change. In (Yakout *et al.*, 2011), the user is consulted to select the updates that are likely to be more beneficial. In (Dallachiesa *et al.*, 2013), confidence scores are not supported. In (Fan *et al.*, 2012), master data is assumed to be available to fix the data. Cost-based models are used by (Bohannon *et al.*, 2005; Cong *et al.*, 2007; Kolahi and Lakshmanan, 2009). However, density-based information is not taken into consideration. That is, they do not consider local structural properties, represented by density information, in the assignment of weights while cleaning the data. The work of (Fan *et al.*, 2014b) interleaves data repairing and record matching. The work is based mainly on the availability of master data and confidence scores assigned by the user. A work by (Chomicki and Marcinkowski, 2005) suggested tuple deletion to repair the violation of FDs instead of modification of attribute values. The model of tuple modification is preferred over the model of tuple deletion as it minimizes the amount of information loss.

Work related to our research on data accuracy and truth discovery from data has been reported previously (Cao *et al.*, 2013; Dong *et al.*, 2009; Wu and Marian, 2011; Yin *et al.*, 2008). The update history of sources is examined in (Dong *et al.*,

2009) to determine the copying relationship between sources in order to discover the true values. In (Cao *et al.*, 2013), accurate values of some attributes are determined through accuracy rules. However, the experimental results indicate that the algorithm is still based mainly on manual user interaction and would perform worse in the absence of master data. In (Wu and Marian, 2011), a framework is proposed that is based on frequency, relevance, and prominence. Scores are assigned to the answers of a query in order to discover the best answer. In (Yin *et al.*, 2008), the discovery of true facts from conflicting information is based on the trustworthiness of data sources, confidence of the facts, and how these facts influence each other.

Work on bag-of-words model, hierarchical clustering, and dimensionality reduction that is related to our research has been reported previously (Dhillon *et al.*, 2002; Slonim and Tishby, 2001; Yang and Pedersen, 1997). In (Slonim and Tishby, 2001), word clustering using an agglomerative algorithm is applied to text classification while in (Dhillon *et al.*, 2002), a divisive algorithm is used for clustering words. Using feature selection, different methods are implemented in (Yang and Pedersen, 1997) to reduce dimensionality. While our work in the discovery of accurate values is similar in spirit to the work that is related to feature selection and agglomerative clustering, it is integrated with the repair of data in a density-based deduplication, a functionality that does not exist in the mentioned work.

# Chapter 4

# Deduplication with Consistency and Accuracy: Experimental Evaluation

This Chapter presents an experimental evaluation of our data cleaning techniques in Chapter 3. In sections 4.2 and 4.3, we conduct qualitative and performance evaluations of our algorithms. In section 4.4, we present a comparative study.

## 4.1 Datasets

Experimental evaluation were conducted to verify the quality and scalability of our algorithms using synthetic and real datasets in various configurations.

| | name | addr | city | phone | type | class |
|---|---|---|---|---|---|---|
| $t_1$ | baja fresh | 3345 kimber dr. | westlake village | 805-498-4049 | mexican | 536 |
| $t_2$ | belvedere the | 9882 little santa monica blvd. | beverly hills | 310-788-2306 | pacific new wave | 537 |
| $t_3$ | benitas frites | 1433 third st. promenade | santa monica | 310-458-2889 | fast food | 538 |
| $t_4$ | bernards | 515 s. olive st. | los angeles | 213-612-1580 | continental | 539 |
| $t_5$ | bistro 45 | 45 s. mentor ave. | pasadena | 818-795-2478 | californian | 540 |
| $t_6$ | brents deli | 19565 parthenia ave. | northridge | 818-886-5679 | delis | 541 |
| $t_7$ | brighton coffee shop | 9600 brighton way | beverly hills | 310-276-7732 | coffee shops | 542 |
| $t_8$ | bristol farms market cafe | 1570 rosecrans ave. s. | pasadena | 310-643-5229 | californian | 543 |
| $t_9$ | brunos | 3838 centinela ave. | mar vista | 310-397-5703 | italian | 544 |
| $t_{10}$ | cafe 50s | 838 lincoln blvd. | venice | 310-399-1955 | american | 545 |

Table 4.1: Instance of *Restaurant* relation

### Customers Synthetic Dataset

A Synthetic dataset, denoted as *Customers*, was used to conduct an evaluation of quality and scalability. We used online data generator tool[1] to generate the synthetic dataset. Initially, we generated 2500 tuples of synthetic data about customers with six attributes including the name, phone number, job title, income level, credit limit, and address with the FD $\phi_{Customers} : IncomeLevel \rightarrow CreditLimit$. $\phi_{Customers}$ is a functional dependency, and asserts that income level determines credit limit.

### Restaurant Real Dataset

We evaluated the quality using a widely used dataset for record matching, namely restaurant dataset[2], denoted as *Restaurant*. We allowed the attribute values that have NULL marks to be part of the data since in reality, some attribute values of a tuple could be missing (NULL). However, we give preference for the attribute values that are not NULL marks over the attributes values of the FDs that are NULL on either side. We also give preference for the attribute values that are not NULL in the accuracy attributes.

[1]`http://www.generatedata.com`
[2]`http://www.cs.utexas.edu/users/ml/riddle`

### Parameters

The experiments were conducted on a computer with an Intel Core i7 CPU (2.00 GHz) and 8GB of RAM. Let $dup_{rate}$ be duplicate rate, i.e., the percentage of duplicate tuples in the data, and $dup_{num}$ be number of duplicates per tuple. Let $e$ be the rate of error, which is the fraction of values of attributes in the duplicates that are modified to have errors. Let $N$ be size of data $D_d$, including the duplicates, $ac$ be number of accuracy attributes, $fn$ be number of FDs, and $ta$ be total number of attributes.

In *Customers* dataset, we ensured that the generated data has no duplicate records and is consistent and accurate by using both an algorithm and a manual check. We considered it to be ground truth, denoted by $D_{g_1}$. We then duplicated some of the tuples and injected errors into $D_{g_1}$ to get the dirty data $D_d$. In *Restaurant* dataset, the schema consists of 6 attributes including name, address, city, phone, type, and class. We postulated the following FD: $\phi_{Restaurant} : phone \rightarrow city$. $\phi_{Restaurant}$ is a functional dependency, and asserts that phone number determines city. The dataset has $N = 864$ tuples including 112 duplicate tuples. Since we are given what tuples are duplicates, we deleted the duplicates to get one tuple per entity. The data values were enclosed in single or double quotations marks. We trimmed these quotations for easier handling as they are not part of the real values of the attributes. We considered the resultant dataset as the ground truth $D_{g_2}$. Table 4.1 illustrates an instance of $D_{g_2}$.

## 4.2 Quality of Cleaning

We used two measures to evaluate the quality of our data cleaning approach for deduplication with data consistency and accuracy, namely *precision* and *recall*. Precision is used to measure the correctness of the cleaning while recall is used to measure the completeness of the cleaning:

- $Precision = \frac{Number\, of\, Correctly\, Identified\, Duplicates}{Number\, of\, Declared\, Duplicates}$
- $Recall = \frac{Number\, of\, Correctly\, Identified\, Duplicates}{Number\, of\, True\, Duplicates}$

Figures 4.1 and 4.2 illustrate the precision and recall experimental results of our data cleaning approach for *Customers* dataset, respectively. We fix $N = 4000$, $dup_{rate} = 30\%$, $dup_{num} = 3$, $ac = 4$, $fn = 1$, $MinPts = 3$, $ta = 6$, $\varepsilon = 0.6$, and $\theta = 0.35$. We varied the rate of error $e$ from 3% to 15%. In Figure 4.1, we measure the precision of the cleaning. We observe that the values of precision start to decline as $e$ starts to increase, as expected. In Figure 4.2, we observe that the values of recall are high with low $e$. As $e$ increases, the values decrease with each increase, as expected.

Figures 4.3 and 4.4 illustrate the experimental results for the precision and recall for *Restaurant* dataset, respectively. We fix $dup_{rate} = 30\%$ and $dup_{num} = 3$. The resultant $N = 1204$ tuples. We also fix $ac = 4$, $fn = 1$, $MinPts = 3$, $ta = 6$, $\varepsilon = 0.6$, and $\theta = 0.35$. We injected errors in the values of the attributes. The rate of error $e$ varied from 3% to 15%. We note that the values of precision and recall generally decline slightly as the rate of error $e$ increases, as expected. The values of precision are relatively high, which means that our algorithms are able to correctly find the vast majority of the declared duplicates. The values of recall are fairly high, which means that our algorithms can capture the majority of true duplicates.

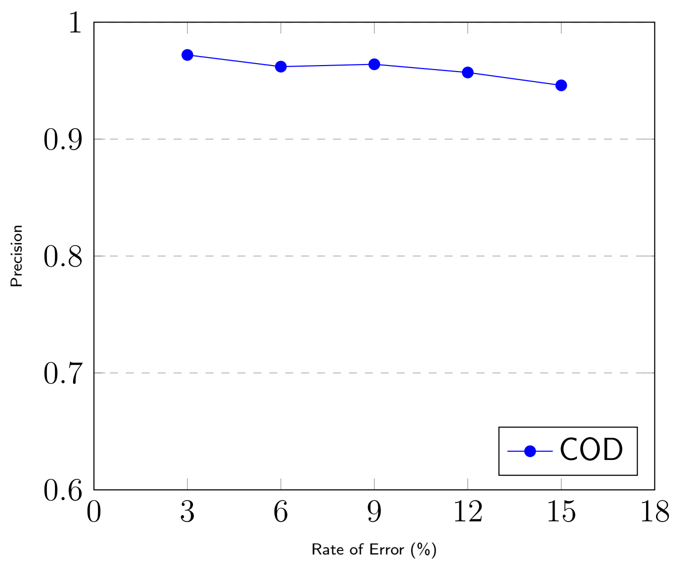

Figure 4.1: Precision at multiple rates of error (*Customers* dataset)

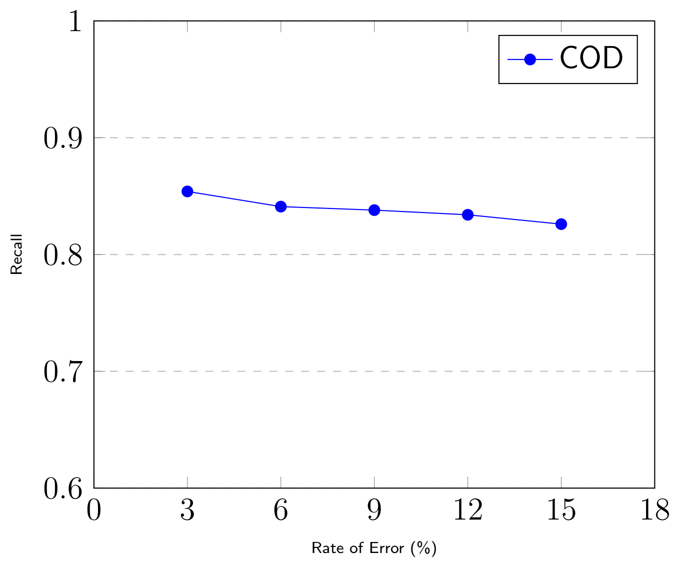

Figure 4.2: Recall at multiple rates of error (*Customers* dataset)

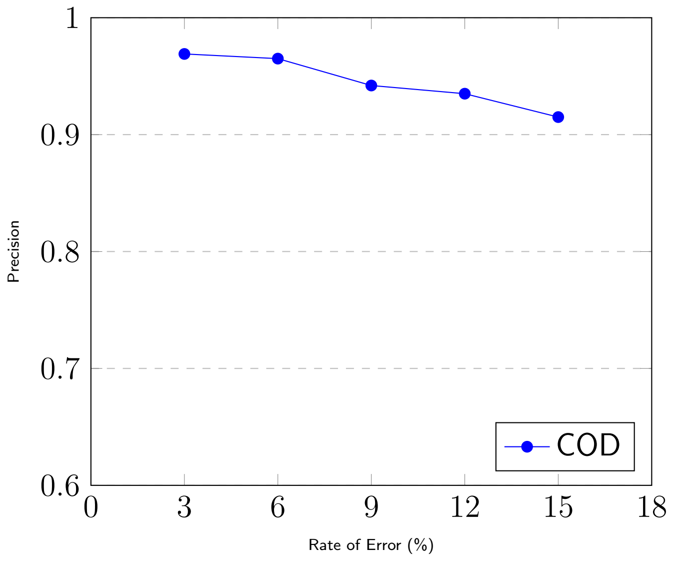

Figure 4.3: Precision at multiple rates of error (*Restaurant* dataset)

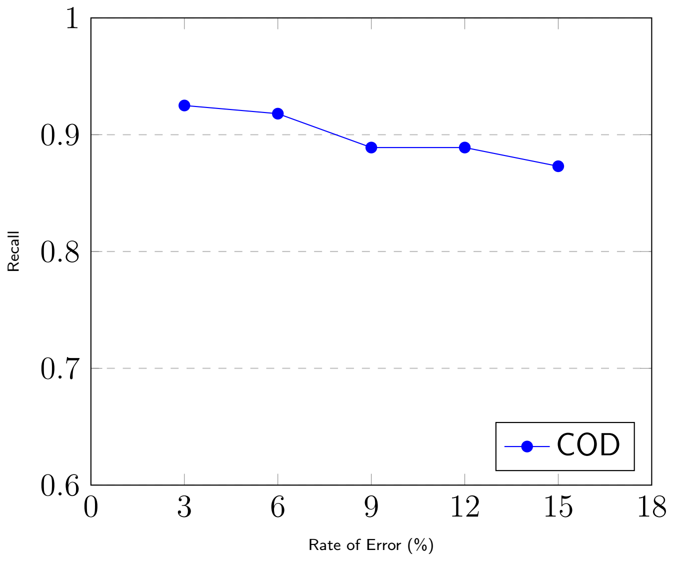

Figure 4.4: Recall at multiple rates of error (*Restaurant* dataset)

## 4.3 Performance of Cleaning

We evaluate experimentally the scalability of data cleaning in terms of running time with respect to number of accuracy attributes, number of FDs, rate of error, number of tuples, and rate of duplicates. *Customers* dataset is used in these experiments.

*1) Scalability w.r.t. the Number of Accuracy Attributes:* In figure 4.5, we studied the scalability with respect to the number of accuracy attributes ($ac$). We fix $N = 4000$, $dup_{rate} = 30\%$, $dup_{num} = 3$, $e = 9\%$, $fn = 1$, $MinPts = 3$, $\varepsilon = 0.6$, and $\theta = 0.35$. We added more attributes of similar domains and data sizes to keep comparable the amount of time to discover the accurate values of each attribute. Having different domains and sizes requires varying times to discover the accurate values which makes the comparison difficult.

As $e$ increases, the running time moderately increases in a relatively linear scale. Adding more accuracy attributes causes more operations in the discoverAccurateData algorithm, but these operations are similar in terms of running time per attribute. This is because the attributes have a fixed number of errors, a similar domain type, and similar data sizes.

*2) Scalability w.r.t. the Number of FDs:* We measured the scalability of our approach against increasing the number of FDs ($fn$). We fix $N = 4000$, $dup_{rate} = 30\%$, $dup_{num} = 3$, $e = 9\%$, $ac = 4$, $MinPts = 3$, $\varepsilon = 0.6$, and $\theta = 0.35$. Figure 4.6 shows the running times for this experiment. We added FDs of similar numbers of attributes, domains, and data sizes to avoid having varying times to repair each FD, as different FDs make the comparison difficult. Comparing to Figure 4.5, the repair of the FDs also grows linearly but more aggressively because of the time needed to process building and handling the classes. This process requires reading the whole dataset to compute the frequencies of the representatives during the cost computations. The total time to complete a repair for an FD by applying the repairData algorithm is higher than the time to discover an accurate value by applying the discoverAccurate-Data algorithm to the attribute values of a cluster. It is still linear because the FDs are of similar numbers of attributes, domains, and data sizes.

*3) Scalability w.r.t. the Rate of Error:* Figures 4.7 illustrates the experimental result of varying rate of error ($e$). We fix $N = 4000$, $dup_{rate} = 30\%$, $dup_{num} = 3$, $ac = 4$, $fn = 1$, $MinPts = 3$, $ta = 6$, $\varepsilon = 0.6$, and $\theta = 0.35$. We notice that the running time of the cleaning increases slightly as we increase $e$. This is due to the fact that varying $e$ does not change number of attributes. This slight increase in the running time is due to the increase in $e$, which makes the per-cluster number of classes to build and handle in the repair increases slightly. In addition, the time required to discover the accurate values is also increased slightly because the number of accuracy attributes is still the same. Thus, varying $e$ in this case does not have significant impact on the performance.

*4) Scalability w.r.t. the Rate of Duplicates:* Figure 4.8 depicts the scalability performance of our approach for various rates of duplicates ($dup_{rate}$). We fix $N = 4000$, $dup_{num} = 3$, $e = 9\%$, $ac = 4$, $fn = 1$, $MinPts = 3$, $ta = 6$, $\varepsilon = 0.6$, and $\theta = 0.35$. We increased the rate $dup_{rate}$ from 15% to 75%. We found that the running time moderately increased. This is expected, as increasing the number of duplicates in the data results in a larger number of clusters to clean which, in turn, requires more time to clean. It is still a moderate increase because $N$, $fn$, and $ac$ are the same.

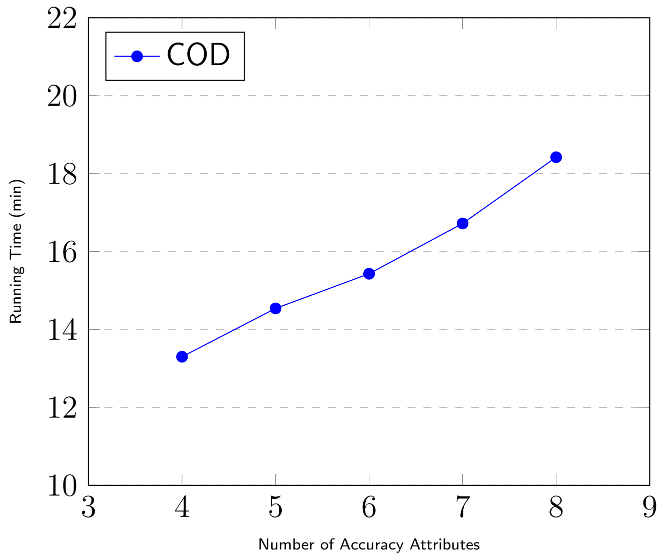

Figure 4.5: Scalability with number of accuracy attributes

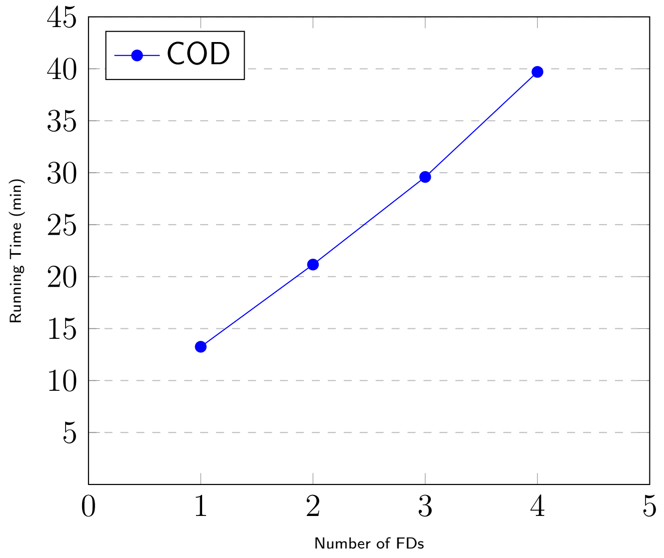

Figure 4.6: Scalability with number of FDs

Comparing to figure 4.7, increasing $dup_{rate}$ has more impact on the performance due to the fact that as $dup_{rate}$ increases, more clusters are formed that require cleaning. Increasing $e$ has less effect on the number of clusters.

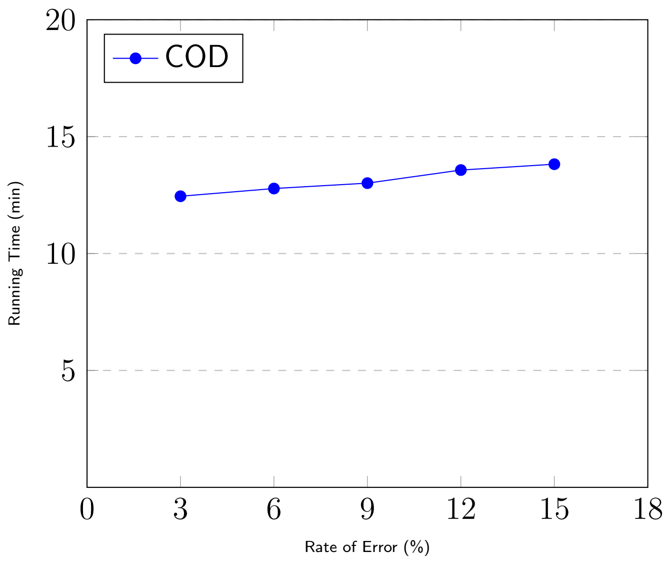

Figure 4.7: Scalability with rate of error

*5) Scalability w.r.t. the Number of Tuples:* In this experiment, we measure the running time of varying the value of $N$ from 2k to 10k. We fix $dup_{rate}$ = 30%, $dup_{num}$ = 3, $e$ = 9%, $ac$ = 4, $fn$ = 1, $MinPts$ = 3, $ta$ = 6, $\varepsilon$ = 0.6, and $\theta$ = 0.35. Figure 4.9 shows that running times grow more rapidly with the increase of $N$. As the number of tuples increases, the duplicate rate $dup_{rate}$ covers more tuples in $N$. This causes an increase in the time required to cluster the duplicate tuples (i.e., clusterTuples algorithm). Also, there are more clusters to clean in the data repair (i.e, repairData algorithm) and discovering of accurate values (i.e., discoverAccurateData algorithm) steps. That is, in data repairing, we have more clusters to clean and a large size of data to deal with per cluster cleaning. In the discovery of accurate values, we have more values to deal with as we have more clusters. The combination of the times required to do the above operations makes the running time grows more rapidly with the size of data $N$.

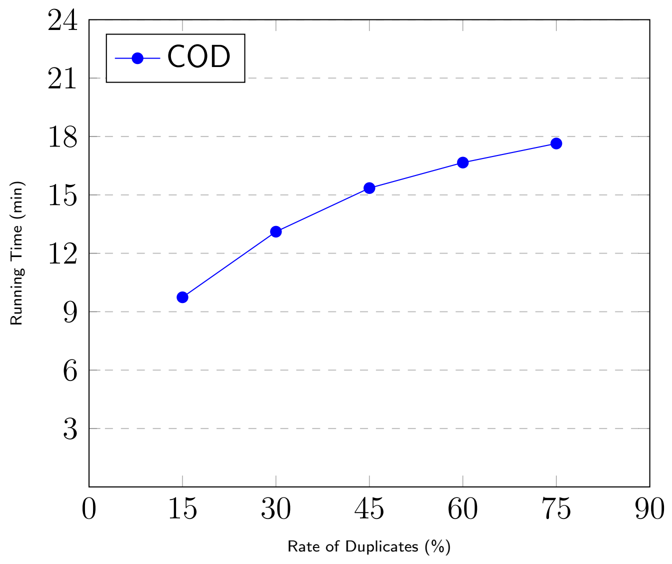

Figure 4.8: Scalability with rate of duplicates

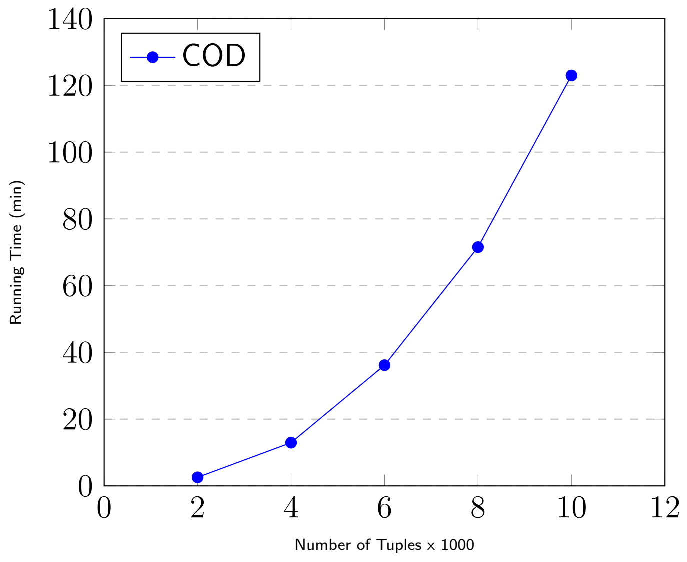

Figure 4.9: Scalability with number of tuples

## 4.4 Comparative Study

In figure 4.10, we compare the quality of our approach to the approach of (Song *et al.*, 2015). They propose an algorithm, denoted FD+QDORC, to perform data clustering and repairing at the same time, with which we compare our algorithm. The objective of performing the simultaneous clustering and repairing is to find a minimum repair in order for the data to be utilized as much as possible, for the purpose of clustering support. Their approach, like ours, is based on the density-based clustering and data repairing using the minimal-based repairing. An existing FD constraint-based repairing technique is combined to perform simultaneous clustering and repairing of integrity constraints. The main difference in our approach is that after the tuples are clustered, the weights are assigned, and then the costs are computed to select the best repair for the FDs. In addition, the FD+QDORC algorithm performs deduplication up to the clustering step, not a complete deduplication process. That is, it does not merge tuples. In their quality results we use for comparison, they used the same *Restaurant* dataset we used. We compare the quality, represented by the recall, of our algorithm COD with their algorithm. The results of the FD+QDORC algorithm were taken from (Song *et al.*, 2015) and have been written manually with careful checking. The values were subtracted from 1 to unify their scale with ours. The repairing errors in their experiment were tested against multiple error rates to show how close the repaired data are to the ground truth. The comparison shows approximated quality results of the two approaches using the same dataset. The two experiments have some differences in the parameter settings, the distance measure, and the quality measure. However, restaurant dataset is a popular dataset of well-known duplicates to evaluate duplicate detection methods, making these differences less significant.

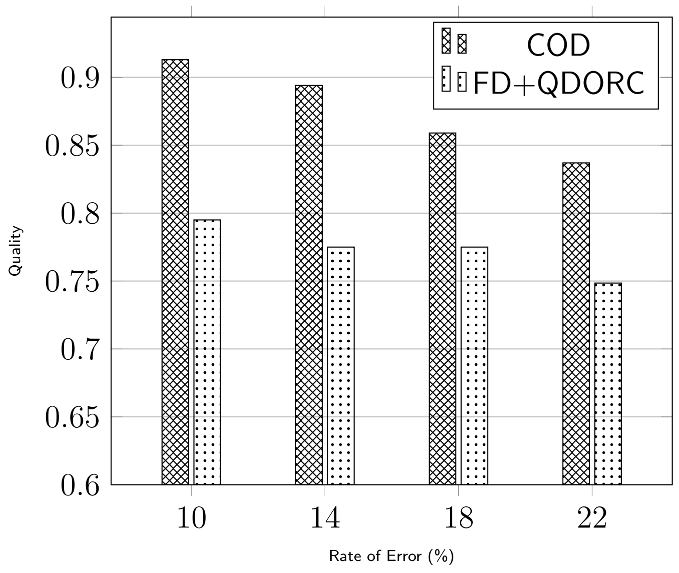

Figure 4.10: The comparison of quality of COD algorithm compared to FD+QDORC algorithm

Our approach achieved better quality results compared to the results in (Song *et al.*, 2015). This shows our algorithm's better quality of fixing errors in the data.

# Chapter 5

# Corroborating Quality of Data Through Density Information

In this Chapter, we present our approach for corroborating quality of data. This Chapter is organized as follows: Sections 5.1 and 5.2 provide an overview of corroborating quality of data through density information. In section 5.3 we present our confidence model. In section 5.4, we present our techniques for corroborating attribute values. In section 5.5, we discuss the merging step, followed by section 5.6, in which we discuss the algorithms. In section 5.7, we present the related work.

## 5.1 Introduction

Data become obsolete over time. According to (Eckerson, 2002), 2% of the data in a customer file may become outdated in one month. This means that 120,000 records of customers in a database of 1,000,000 records could be stale in six months, that is, 12% of the total records. (Redman, 1998) states that 40-60% of the expenses of the service organization may be consumed as a result of unclean data. Reports and research listed by (Marsh, 2005) indicate that cost stemming from unclean data is identified in 75% of the organizations and business intelligence projects often fail because of unclean data. Also 33% of the organizations canceled or delayed new IT systems due to unclean data. These emphasize the importance of data cleaning.

In practice, data in databases become stale or inaccurate over time, that is, attribute values in the databases may not represent the up-to-date values of the real-world entities they are referring to. Data quality issues arise when integrating data from multiple sources (Dasu and Johnson, 2003). Duplicate records may happen because of the integration of multiple data sources where the same entity is entered differently in each source. Also, duplicates may appear within a single database. These different representations of the same entity are caused by reasons such as different times of data entry or different data entry requirements (Naumann and Herschel, 2010). Errors could be also introduced in duplicate records as the result of transcription errors and lack of standard formats (Elmagarmid *et al.*, 2007). Temporal databases provide support for data involving time, however, in real-life, it is not enough to depend only on timestamps as they could be imprecise or unknown (Zhang *et al.*, 2010). A model was proposed by (Fan *et al.*, 2011) where data currency is derived in terms of currency orders and currency constraints. Temporal ordering of records was proposed in the case where the timestamps are absent and no information is given about the accuracy of data. However, the attribute values may not provide enough temporal information to fix all the data.

| | **CompName** | **RepName** | **Position** | **TotOrders** | **TechSuport** | **DiscRate** |
|---|---|---|---|---|---|---|
| $t_1$ | Ocean Tech | Sara Mark | Vice President | 320 | Advanced | 10 |
| $t_2$ | Nipton Placement | Rami Ramsy | Marketing Associate | 2100 | Advanced | 5 |
| $t_3$ | Rocks Ltd | Christophet Salme | Manager | 144 | Advanced | 10 |
| $t_4$ | Rocks | Christophet Salem | Manager | NULL | Advanced | 5 |
| $t_5$ | Rocks | Christopher Salem | Assistant Manager | 188 | Intermediate | 12 |
| $t_6$ | Rocks Ltd | Christopher Salem | Employee | 144 | Intermediate | 12 |
| $t_7$ | Rocks Ltd | C. Salme | Assistant Manager | 188 | Intermediate | 10 |
| $t_8$ | Mars Equipment | Amy Ford | Salesperson | 77 | Basic | 2 |
| $t_9$ | Rocks | Ramond Jack | Engineer | NULL | Advanced | 10 |
| $t_{10}$ | Spark Ltd | jack Neil | NULL | 200 | Advanced | 12 |
| $t_{11}$ | Alpha inc | Adam Blake | CAD Draftsperson | 309 | Basic | 2 |

Table 5.1: Instance of *Clients* relation

A work by (Fan *et al.*, 2014a) for conflict resolution combines data currency (Fan *et al.*, 2011) and data consistency (Fan *et al.*, 2008) techniques, but is based mainly on manual user interaction to identify a single tuple for a set of tuples pertaining to the same entity. There has been work on data accuracy and truth discovery from data that has been either based mainly on manual user interaction (Cao *et al.*, 2013), or it does not address the problem of duplicate elimination (Wu and Marian, 2011). Approaches in deduplication (Cohen and Richman, 2002; Hernández and Stolfo, 1998; Ravikumar and Cohen, 2004; Sarawagi and Bhamidipaty, 2002) require either training data, manual user interaction, or they do not address the problem of data currency in duplicate elimination. These make it challenging to discover the current or accurate values in order to merge the tuples of a cluster into a single representative tuple. In this thesis, we argue that we could use the density of data to corroborate the trustworthiness of the attribute values, that is, to provide more evidence to be taken into consideration upon eliminating duplicates, given the currency orders and currency constraints of (Fan *et al.*, 2011). In the following example, we illustrate these challenges.

*Example 5.1*: Consider table 5.1 that represents data about clients integrated from various data sources across some branches of a company. The attributes of the schema include the name of the company (**CompName**), name of the representative of the company (**RepName**), position of the representative (**Position**), total orders that were ordered by the client (**TotOrders**), type of technical support provided to the client (**TechSupport**), and the discount rate given to the client (**DiscRate**). Consider also four constraints: the first constraint states that an employee may be promoted from “Assistant Manager” to “Manager”. The second constraint states that the total number of orders is increasing over time. The third one states that a technical support type is promoted over time from “Intermediate” to “Advanced”. The fourth constraint states that the discount rate is increasing over time. In terms of currency orders, there are four. The first one indicates that a value of “Manager” is more current than a value of “Assistant Manager”. The second one indicates that for different total orders numbers, a greater one is more current than a smaller one. The third one indicates that a technical support of type “Advanced” is more current than a type of “Intermediate”. The fourth one indicates that for different discount rates, a higher one is more current than a smaller one. By the tuple matching and clustering techniques of the duplication elimination process, we discover that tuples $t_3$-$t_7$ represent the same real-world entity, that is, they are duplicate records in the table so they are grouped in one cluster.

In order to merge the duplicate tuples into a single tuple, we need to find the correct values of the attributes in the tuples, that is, we want to find the values that are current and accurate. However, some of the attribute values in the cluster may be outdated and inaccurate. In this thesis, we differentiate between inaccurate values that are caused by transcription errors and lack of standard formats, such as writing an address in different formats, and the inaccurate values that are caused over time, but were once correct, such as moving into new address. We consider the first type of data quality issue as data inaccuracy, while we consider the second type as outdated data.

By the first currency order, we may tell that the values in tuples $t_3$[Position] and $t_4$[Position] are more current than the values of $t_5$[Position], and $t_7$[Position]. By the second currency order we may tell that the values of $t_5$[TotOrders] and $t_7$[TotOrders] are more current than the values of $t_3$[TotOrders] and $t_6$[TotOrders]. The value of $t_4$[TotOrders] is unknown. By the third currency order we may tell that the values of $t_3$[TechSupport] and $t_4$[TechSupport] are more current than the values of $t_5$[TechSupport], $t_6$[TechSupport], and $t_7$[TechSupport]. By the fourth currency order we may tell that the values of $t_5$[DiscRate] and $t_6$[DiscRate] are more current than the values of $t_3$[DiscRate], $t_4$[DiscRate], and $t_7$[DiscRate].

However, the value of $t_6$[Position] is “Employee”. It is still not determined whether it is more current than “Manager” or not as there is no currency information about this value. Also, there are still values in the attributes **CompName** and **RepName** which have no currency information. In the attribute **CompName**, there are two distinct attribute values including “Rocks Ltd” and “Rocks”. These differences could be caused by a lack of standard formats in writing the name of the company. There is not clear evidence in the data to tell which value is more accurate to choose. In the attribute **RepName**, there are four distinct values including “Christophet Salme”, “Christophet Salem”, “Christopher Salem”, and “C. Salme”. These different values could be caused by transcription errors and a lack of standard formats. None of these two attributes have defined currency information to decide which values are the most current or accurate. Decisions about the final single tuple for the cluster cannot be taken without having the correct values for these attributes.

If we can use the density information to corroborate which of the above values are more trustworthy than others, we will be able to identify a single tuple in which each attribute value has the latest and most accurate value in the set of tuples referring to the same entity. We can obtain the density information associated with a tuple by taking into consideration the density of tuples in a region within a specified distance around this tuple. In figure 3.1, for instance, for a minimum number of data points, i.e. tuples, of 3 that is needed to form a dense region within a distance $\varepsilon$. Tuple $p_1$ is in higher density region because there are three tuples including $p_1$ within $\varepsilon$, compared to $p_4$ that has only two tuples within $\varepsilon$.

Returning to example 5.1, if we use density-based clustering, we could discover that tuples $t_3$, $t_5$, and $t_7$ lie in a high density region, while tuples $t_4$ and $t_6$ lie in a lower density region, so we may trust the values of $t_3$, $t_5$, and $t_7$ more. Thus, we could identify that 'Rocks Ltd", "Christopher Salem" and "Manager" have more trust than others, and then conclude that "Rocks Ltd", "Christopher Salem", "Manager", "188", "Advanced", "12" are the most accurate and latest values in the set of tuples referring to the same entity.

We propose an approach that places trustworthiness scores for the attribute values where data cleaning is achieved without a need for human effort, master data, or training data. We make the following contributions:

- A corroborating model for attribute values based on confidence scores of tuples and the density of the data where the tuples in the dense regions are packed together. The confidence model of the tuples is represented as confidence scores about the correctness of the tuples w.r.t. the currency orders and currency constraints.

- We introduce a data cleaning framework and algorithms that determine the current and accurate data during the deduplication process without manual user interaction. It discovers the correct values based on the trustworthiness of the attribute values for the erroneous attributes values that do not have necessarily currency information in terms of currency orders and currency constraints.

- We conduct various experiments that show the scalability and quality of our proposed framework and algorithms. We use synthetic and real-world datasets. We show that our model can effectively corroborate the trustworthiness of attributes values.

## 5.2 Density-based Partitioning

We use the same clustering algorithm that does not require determination of the number of clusters in advance, that is, DBSCAN. This feature conforms to the problem of duplicate identification where the number of real-world entities is not known a priori. DBSCAN algorithm has been discussed earlier in Chapters 2 and 3. Returning to example 5.1, DBSCAN may cluster the tuples with $MinPts = 3$ and $\varepsilon = 0.6$ as follows: $t_3$, $t_5$, and $t_7$ as core tuples; $t_4$ and $t_6$ as border tuples; and $t_1$, $t_2$, and $t_8$-$t_{11}$ as outliers.

## 5.3 Confidence Model

The decision about the currency of the attribute values of the tuples per cluster is based on a confidence technique that measures the currency of tuples. Let $R$ be a relation schema of $D$ consisting of a set $T$ of $h$ attributes, denoted $\{A_1, A_2, ..., A_h\}$.

Let $Z$ be a set of $w$ clusters $\{C_1, C_2, ..., C_w\}$ that are clustered according to the density-based clustering algorithm where each cluster is a set of distinct tuples in $D$. Let $I_c$ be an instance of the tuples of a singleton set $E$ of cluster $C \in Z$. Let $R_1$ be a relation schema consisting of a set $V \subset T$ of $m$ attributes, denoted $\{A_1, A_2, ..., A_m\}$, that is, $R_1 = \prod_V(I_c)$, in which each $A_i$ has a domain $dom_{A_i}$. We explain the data currency that is specified in terms of currency orders and currency constraints (adapted from (Fan *et al.*, 2011, 2014a)). Let $I_r$ be an instance of $R_1$. Let $I_t$ be a temporal instance of $I_r$ that is given by $(I_r, \preccurlyeq_{A_1}, ..., \preccurlyeq_{A_m})$, where each of $\preccurlyeq_{A_i}$ is partial order that is defined on $I_r$ such that for two tuples $t_1$,$t_2 \in I_r$, $t_1 \preccurlyeq_{A_i} t_2$ if and only if either $t_2[A_i]$ is more current than $t_1[A_i]$, or $t_1[A_i] = t_2[A_i]$. The partial order $\preccurlyeq_{A_i}$ is also referred to as a currency order for attribute $A_i \in I_r$. In the currency order, a missing value of an attribute, i.e. NULL, is ranked the lowest. Semantics of data can be used to derive information. Currency constraint $\varphi$ can be written in the form: $\forall t_1, t_2 : R(\psi \rightarrow t_1 \prec_{A_k} t_2)$, where $\psi$ is a conjunction of predicates that takes the form: (1) $t_1 \prec_{A_f} t_2$, that is, in $A_f$, $t_2$ is more current than $t_1$; (2) $t_1[A_f]$ $\mathsf{oper}$ $t_2[A_f]$; and (3) $t_1[A_f]$ $\mathsf{oper}$ $c$ (4) $t_2[A_f]$ $\mathsf{oper}$ $c$, where $c$ is a constant and $\mathsf{oper}$ is $\geq$, $\leq$, $=$, $\neq$, $>$, or $<$.

While our method in computing the confidence scores from the current data is similar in spirit to the technique in (Fan *et al.*, 2011), it is used to combine broader data quality aspects including data currency and data accuracy for duplicate elimination. This is based on common sense and intuition that data density may enable us to be more confident about the level of the correctness of the attribute values, as there are regions with higher density than others. It provides more information to corroborate attribute values and increase certainty of the most current tuples in the same cluster which, may help in the cleaning of other quality issues such as data accuracy as well.

That is, accuracy of data is also discovered for the attributes that have no currency orders and currency constraints. Besides, we do not interact with the users to solicit additional data currency information during the cleaning process as it is expensive for users to provide information manually. In addition, no timestamps are assumed to be available.

We need to increase certainty of the tuples that may have the correct values by utilizing currency orders and currency constraints. The tuple with the most current values has more confidence in the correctness of its attribute values, and thus has a higher confidence score. We follow previous work (Bohannon *et al.*, 2005) and assume that there is a score that reflects the confidence in the correctness of the values of the tuple. The confidence score is associated with a tuple $t$ for each tuple $t \in I_r$. Let $t[A_i]$ denote the most current value of an attribute $A_i$ in a tuple $t$, and let $M_{c_t}$ be the number of the most current attribute values in the tuple $t$. There could be more than one most current value for an attribute as there could be more than one partial currency order in an attribute. Recall that $V$ is a set of attributes that have currency orders and currency constraints on them. We define the confidence of a tuple, where a higher score indicates more confidence, as:

**Definition 8.** *Confidence of a tuple $t \in I_r$, denoted by $c(t)$, is the fraction of the most current attribute values provided by $t$.*

$$c(t) = \frac{M_{c_t}}{|V|}$$

*Example 5.3*: Assume that the tuples of table 5.1 are clustered as in example 5.1, that is, $t_3$, $t_4$, $t_5$, $t_6$, and $t_7$ form a single cluster. Consider the following currency constraints:

- $\psi_1$: $\forall a, b$: *Clients*($a$[Position] = "Manager" $\wedge$ $b$[Position] = "Assistant Manager") $\rightarrow b \prec_{position} a$)
- $\psi_2$: $\forall a, b$: *Clients*($a$[TotOrders] $>$ $b$[TotOrders] $\rightarrow b \prec_{TotOrders} a$)
- $\psi_3$: $\forall a, b$: *Clients*($a$[TechSupport] = "Advanced" $\wedge$ $b$[TechSupport] = "Intermediate") $\rightarrow b \prec_{TechSupport} a$)
- $\psi_4$: $\forall a, b$: *Clients*($a$[DiscRate] $>$ $b$[DiscRate] $\rightarrow b \prec_{DiscRate} a$)

$\psi_1$ is a currency constraint asserts that the position of "manager" is more current than "assistant manager". This comes from the fact that an employee in a company is promoted to a higher position. $\psi_2$ is a currency constraint which asserts that having a greater number of orders lends more currency. This comes from the fact that number of transactions typically increases. $\psi_3$ is a currency constraint which asserts that a technical support level of "Advanced" is more current than a level of "Intermediate". This comes from the company's policy that states that technical support of a client is changed from a lower level such as "Intermediate" to a higher level such as "Advanced". $\psi_4$ is a currency constraint which asserts that having a greater discount rate lends more currency. This comes from the company's policy that states that clients are given more discounts over time.

- From $\psi_1$, we determine that $t_3$[Position] and $t_4$[Position] are more current
- From $\psi_2$, we determine that $t_5$[TotalOrders] and $t_7$[TotalOrders] are more current
- From $\psi_3$, we determine that $t_3$[TechSupport] and $t_4$[TechSupport] are more current

- From $\psi_4$, we determine that $t_5$[DiscRate] and $t_6$[DiscRate] are more current
- $V = 4$. We compute the confidence scores for $t_3$, $t_4$, $t_5$, $t_6$, and $t_7$
  - $c(t_3) = 2/4 = 0.5$
  - $c(t_4) = 2/4 = 0.5$
  - $c(t_5) = 2/4 = 0.5$
  - $c(t_6) = 1/4 = 0.25$
  - $c(t_7) = 1/4 = 0.25$
- We determine that tuples $t_3$, $t_4$, and $t_5$ have greater confidence than tuples $t_6$ and $t_7$.

## 5.4 Corroborating of Attribute Values

In example 5.3, we determined the confidence scores about the correctness of the tuples from the currency orders. However, there are other attributes for which we do not have enough currency orders and currency constraints to make decisions about their accurate values. Now, we can bring data currency and data accuracy together. We consider the currency orders, and the fact that there are two and four distinct values in **CompName** and **RepName**, respectively. We also consider the fact that it is not certain yet whether a value of “Manager” or “Employee” value in **Position** is more current. We end up with 16 potential single tuple candidates:

- (“Rocks Ltd” , “Christopher Salme” , “Manager”, “188” , “Advanced” , “12”)
- (“Rocks Ltd” , “Christophet Salem” , “Manager”, “188” , “Advanced” , “12”)

- (“Rocks Ltd” , “Christopher Salem” , “Manager”, “188” , “Advanced” , “12”)
- (“Rocks Ltd” , “C. Salme” , “Manager”, “188” , “Advanced” , “12”)
- (“Rocks” , “Christopher Salme” , “Manager”, “188” , “Advanced” , “12”)
- (“Rocks” , “Christophet Salem” , “Manager”, “188” , “Advanced” , “12”)
- (“Rocks” , “Christopher Salem” , “Manager”, “188” , “Advanced” , “12”)
- (“Rocks” , “C. Salme” , “Manager”, “188” , “Advanced” , “12”)
- (“Rocks Ltd” , “Christopher Salme” , “Employee”, “188” , “Advanced” , “12”)
- (“Rocks Ltd” , “Christophet Salem” , “Employee”, “188” , “Advanced” , “12”)
- (“Rocks Ltd” , “Christopher Salem” , “Employee”, “188” , “Advanced” , “12”)
- (“Rocks Ltd” , “C. Salme” , “Employee”, “188” , “Advanced” , “10”)
- (“Rocks” , “Christopher Salme” , “Employee”, “188” , “Advanced” , “12”)
- (“Rocks” , “Christophet Salem” , “Employee”, “188” , “Advanced” , “12”)
- (“Rocks” , “Christopher Salem” , “Employee”, “188” , “Advanced” , “12”)
- (“Rocks” , “C. Salme” , “Employee”, “188” , “Advanced” , “12”)

Still, we need to decide which attribute values are more accurate among the candidates. Let $R_2$ be a relation schema consisting of a subset $U \subset T$ of $n$ attributes, denoted by $\{A_1, A_2, ..., A_n\}$. That is, $R_2 = \prod_U(I_c)$, in which each $A_j$ has a domain $dom_{A_j}$. Let $I_a$ be an instance of $R_2$. Let $P$ be a set of equivalence classes of an attribute $A_j$ that contains the tuples that share the same value $t[A_j]$ in P. Let $p_j$ be a representative of an equivalence class in $P$. Let $K$ and $B$ be two sets of core and border tuples, respectively. Let $c_p$ be the number of times a representative $p_j$ exists in each core tuple $t \in K$, and let $b_p$ be the number of times the representative $p_j$ exists in each border tuple $t \in B$. Let $c_n$ and $b_n$ be the numbers of core and border tuples, respectively, in $I_a$. We define the weight of a representative as:

**Definition 9.** *Weight of representative, denoted by $w(p_j)$, is the summation of the fractions of how many times the representative exists in the core and border tuples $\in I_a$.*

$$w(p_j) = \frac{c_p}{c_n} + \frac{b_p}{b_n}$$

*Example 5.4.1*: We compute the weight of the attributes: **CompName** and **RepName**. The core tuples are $t_3$, $t_5$, and $t_7$. The border tuples are $t_4$ and $t_6$.

- There are two representatives in the attribute **CompName**: $p_{CompName_1}$ and $p_{CompName_2}$. They represent the distinct values: "Rocks Ltd" and "Rocks", respectively.

- There are four representatives in the attribute **RepName**: $p_{RepName_1}$, $p_{RepName_2}$, $p_{RepName_3}$, and $p_{RepName_4}$. They represent the distinct values: "Christophet Salme", "Christophet Salem", "Christopher Salem", and "C. Salem", respectively.

- There are two representatives in the attribute **Position**: $p_{Position_1}$, $p_{Position_2}$. They represent the distinct values "Manager" and "Employee", respectively.

- We compute the weight of each representative

  - $w(p_{CompName_1}) = 2/3 + 1/2 = 1.166$

- $w(p_{CompName_2}) = 1/3 + 1/2 = 0.833$
- $w(p_{RepName_1}) = 1/3 + 0 = 0.333$
- $w(p_{RepName_2}) = 0 + 1/2 = 0.5$
- $w(p_{RepName_3}) = 1/3 + 1/2 = 0.833$
- $w(p_{RepName_4}) = 1/3 + 0/2 = 0.333$
- $w(p_{Position_1}) = 1/3 + 1/2 = 0.833$
- $w(p_{Position_2}) = 0 + 1/2 = 0.5$

Now, we introduce the trustworthiness of a tuple. Let $r_n$ be the number of times a representative $p_j$ exists in an attribute value in $I_c$. We have from section 5.3, the confidence scores to determine which tuples have more confidence in their values. We define the trustworthiness of a tuple, where a higher score indicates more trust in the attribute value, as:

**Definition 10.** *Trustworthiness of a representative $p_j$, denoted by $u(p_j)$, is the summation of the confidence scores of the tuples where it exists in $I_c$ and its weight.*

$$u(p_j) = \sum_{i=1}^{r_n} c(t) + w(p_j)$$

In this model, a missing value of an attribute, i.e. NULL, is ranked the lowest. The following example illustrates how the correct values of final single tuple are found.

*Example 5.4.2*: We compute the trust of the remaining three attribute values of table 5.1, namely **CompName**, **RepName**, and **Position**. We have the confidence scores and the weights from examples 5.3 and 5.4.1, respectively.

- We compute the trust score for the representatives

  - $u(p_{CompName_1}) = (0.5 + 0.25 + 0.25) + 1.166 = 2.166$
  - $u(p_{CompName_2}) = (0.5 + 0.5) + 0.833 = 1.833$
  - $u(p_{RepName_1}) = 0.5 + 0.333 = 0.833$
  - $u(p_{RepName_2}) = 0.5 + 0.5 = 1$
  - $u(p_{RepName_3}) = (0.5 + 0.25) + 0.833 = 1.583$
  - $u(p_{RepName_4}) = 0.25 + 0.333 = 0.583$
  - $u(p_{Position_1}) = (0.5 + 0.5) + 0.833 = 1.833$
  - $u(p_{Position_2}) = 0.25 + 0.5 = 0.75$

- Since $u(p_{CompName_1})$, $u(p_{RepName_3})$, and $u(p_{Position_1})$ have the highest trust scores, we conclude that $u$(“Rocks Ltd”), $u$(“Christopher Salem”), and $u$(“Manager”) are the most accurate values, respectively.

## 5.5 Tuple Merging

Now, we have the correct attribute values that are required to identify the final single tuple which represents the entity that has duplicates. It is: (“Rocks Ltd” , “Christopher Salem” , “Manager”, “188” , “Advanced” , “12”). This represents in our approach the final task of the duplicate elimination process, that is, the merge task.

## 5.6 Corroborating Algorithm

This section describes the steps of our data cleaning algorithm, referred to as CURET (**C**orroborating Q**u**ality of Data Th**r**ough D**e**nsity Informa**t**ion). The algorithm clusters the data using density based clustering (line 2), which returns a set of core tuples $K$, a set of border tuples $B$, a set of outliers $L$, and a set of clusters $Z$. The algorithm starts cleaning each cluster by processing each attribute in each cluster $C$ (lines 3-4).

**Algorithm 5** CURET

```
Input:  dirty data $D$
        set of currency constraints $\Gamma$
        temporal instance $I_t$
        set $V$ of currency attributes: $V \subset T$
        set $U$ of accuracy attributes: $U \subset T$
        radius of neighborhood $\varepsilon$
        minimum number of points $MinPts$
Output: clean data $D'$
 1: $moCur$, $conf$, $repInfo$, $singleTuple$ = [ ], [ ], [ ], [ ]
 2: $K$, $B$, $L$, $Z$ = clusterBasedOnDensity($D$, $MinPts$, $\varepsilon$)
 3: for each cluster $C \in Z$ do
 4:   for each attribute $A_i$ in $V$ do
 5:     $moCur$.append(inferMostCurr($C$, $\Gamma$, $I_t$, $A_i$))
 6:   for each tuple $t \in C$ do
 7:     $conf$.append(count($moCur$: $moCur \in t$)/$|V|$)
 8:   for each attribute $A_j$ in $U$ do
 9:     $repInfo$.append(getRepInfo($C$, $\Gamma$, $A_j$, $K$, $B$))
10:   $corVal$ = corroborateTrust($C$, $repInfo$, $conf$, $U$)
11:   $singTup$ = $corVal$ + ($moCur$ - ($corVal \cap moCur$))
12:   insert $singTup$ into $D'$
13: insert $L$ into $D'$
14: return $D'$
```

**Algorithm 6** getRepInfo($C$, $\Gamma$, $A_j$, $K$, $B$)

```
1:  $repInfo = [\,]$; $w_o, w_b = 0, 0$
2:  $c_n$ = findNumberOfCoreTuples($K$, $C$)
3:  $b_n$ = findNumberOfBorderTuples($B$, $C$)
4:  $repList$ = getRepresentatives($C$, $\Gamma$, $A_j$, $K$, $B$)
5:  **for each** representative $p \in repList$ **do**
6:    **for each** tuple $t \in C$ **do**
7:      **if** $p \in K$ **then**
8:        increment($w_o$) $/*\, increment\, by\, 1 */$
9:      **else**
10:       increment($w_b$) $/*\, increment\, by\, 1 */$
11:   $weight = w_o/|C| + w_b/|C|$
12:   $repInfo$ = markWeight($weight$, $repList$)
13: **return** $repInfo$
```

A list (*moCur*) of the most current values is created for each attribute using the currency orders and currency constraints (line 5). In lines (6-7), the confidence score of each tuple in the cluster is computed, using the information in the list (*moCur*) and the technique in section 5.3. Then, the confidence of each tuple is added to a list (*conf*). For the attributes that have more than one current value or have values with no currency orders and currency constraints to infer from, the algorithm starts in line 9 discovering the representatives for each attribute, and saves this information in a list ($repInfo$).

In line 10, it corroborates the most trustworthy attribute values. In line 11, we merge the attribute values of $corVal$, that is, the attribute values that have transcription errors, errors caused by lack of standard formats, or values that have no complete currency information, with the values of *moCur*. That is, the attribute values that have either complete or incomplete currency information.

**Algorithm 7** corroborateTrust($C$, $repInfo$, $conf$, $U$)

```
1: $trustWorth = [\,]$
2: for each attribute $A_h$ in $U$ do
3:   for each representative $p$ in $repInfo$ do
4:     $weight$ = getWeight($repInfo$, $C$)
5:     $trustWorth$.append($\sum conf(p) + weight$)
6:   $corVal$.append(maximum($trustWorth$))
7: return $corVal$
```

We exclude from *moCur* the attributes values that have no complete currency information because we already corroborated the trustworthiness of their values in $corVal$. Discovering the final single value for each attribute for each cluster means that we have finished identifying a single tuple for each set of duplicate tuples referring to the same real-world entity. In line 12, we insert the final single tuple into $D'$. In line 13, the list of outliers, i.e. non-duplicate tuples, is inserted into $D'$ to get the final clean data $D'$ that have a single tuple for each cluster $C$.

In algorithm 6, the numbers of core and border tuples in each cluster are found (lines 2-3). Then, the representatives of the attribute $A_j$ are found and saved in a list (*repList*). In lines 5-12, the weight of each representative is found. The algorithm checks whether the representative is in the set of core tuples or border tuples, finds how many times a representative exists in $K$ and $B$, then it computes the final weight (line 11). In line 12, the algorithm marks each representative with its weight and saves the results in a list $repInfo$, then it is returned (line 13).

Algorithm 7 returns the most trustworthy representative among the representatives of the attribute values of $U$, according to the trustworthiness definition in section 5.4. In line 4, it gets the weight of the representative, then it computes the trustworthiness of the representative of an attribute $A_h$ (line 5). In line 6, it selects the most trustworthy representative by selecting the one with the highest score.

## 5.7 Related Work

There has been a host of work on temporal databases. Temporal databases provide support for data involving time including transaction time, valid time, or both (refer to (Chomicki and Toman, 2005) for a survey). They assume the availability of timestamps in which it is possible to decide that a tuple is more current than another. However, in real life, it is not enough to depend only on timestamps as they could be unknown or imprecise (Zhang *et al.*, 2010). A model is proposed by (Fan *et al.*, 2011) where data currency is modeled in terms of currency orders and currency constraints. Temporal ordering of records is derived in the case that the timestamps are absent. However, assuming the absence of reliable timestamps, use of temporal ordering might not lead to a better deduplication if the attribute values do not provide enough information to derive the temporal ordering. A work by (Fan *et al.*, 2014a) for conflict resolution combines data currency (Fan *et al.*, 2011) and data consistency (Fan *et al.*, 2008) techniques. Data currency is specified in terms of currency orders and currency constraints. Data consistency is specified in terms of CFDs. The conflict resolution is based on interleaving inferences of data currency and data consistency, and on manual user interaction, to identify a single tuple for a set of tuples pertaining to the same entity. That is, some attribute values are solicited from the user to help in conflict resolution. Although the technique of (Fan *et al.*, 2014a) may still work without manual user interaction, the quality of their results decreases without it.

There has been work on data accuracy and truth discovery from data (Cao *et al.*, 2013; Dong *et al.*, 2009; Wu and Marian, 2011). The work of (Cao *et al.*, 2013) was based mainly on manual user interaction and a need for master data. The work of (Dong *et al.*, 2009) examines the update history of sources and determines the copying relationship between sources to improve the quality of the integrated data. In (Wu and Marian, 2011), a framework is proposed to aggregate the relevant query results of a search engine from multiple sources. The algorithm assigns scores based on the frequency of the answer, relevance of the originality of the pages, and the prominence of the answer. However, the work of (Dong *et al.*, 2009) does not address data currency problems, while the works of (Cao *et al.*, 2013; Wu and Marian, 2011) do not address the problem of duplicate elimination and data currency.

Work related to our research on duplicate detection has been reported previously (refer to (Elmagarmid *et al.*, 2007; Herzog *et al.*, 2007) for surveys). Those required either training data for supervised learning (Cohen and Richman, 2002; McCallum and Wellner, 2004), or manual user interaction for active learning-based techniques (Sarawagi and Bhamidipaty, 2002). Other approaches such as unsupervised learning techniques (Ravikumar and Cohen, 2004) and distance-based techniques (Chaudhuri *et al.*, 2005; Hernández and Stolfo, 1998) do not address the problem of data currency in duplicate elimination. A work by (Song *et al.*, 2015) performs data clustering and repairing over unclean data simultaneously where the clustering is a density-based clustering. However, it does not address data currency nor does it merge tuples. Temporal records may help in record linkage as shown by the work of (Li *et al.*, 2012). They introduced a framework to identify tuples that describe the same entity over time. They also proposed clustering methods that considered time order of records. However, they assume the availability of timestamps.

We believe that our approach would be useful for practical cases in duplicate records when there is not enough information regarding currency orders and currency constraints or there are errors in the data resulting from transcription errors and lack of standard formats. Furthermore, our technique complements the approaches developed in (Fan *et al.*, 2014a, 2011).

# Chapter 6

# Corroborating Quality of Data: Experimental Evaluation

In this Chapter, we run experiments to determine the scalability and the quality of our data cleaning techniques in Chapter 5. In sections 6.2 and 6.3, we conduct scalability and qualitative evaluations of our algorithms. In section 6.4, we compare our algorithm to the algorithm introduced in (Fan *et al.*, 2014a).

## 6.1 Datasets

We present an experimental study of our data cleaning approach using real and synthetic datasets in various configurations.

### Census Income Real Dataset

We used a real dataset, namely *Census Income*, from UCI machine learning repository[1]. It contains data records about US citizens that describe attributes related to their income (more than $50K or less). For our experiments, we used the attributes: **age**, **education**, **marital-status**, **occupation**, **relationship**, **native-country**, and **income**. There are also missing values, and no reliable timestamps are available in this dataset. We chose randomly 600 tuples from this dataset, and we checked them by using an algorithm to ensure there are no duplicates. We considered these tuples to be the ground truth. We derived 123 currency constraints. We use this dataset to test the quality of our techniques.

### Clients Synthetic Dataset

We used a synthetic dataset, namely *Clients*, that was generated using an online tool[2]. Its schema adheres to the schema of table 5.1. In reality, some attribute values of a tuple could be missing (NULL), so we allowed the attribute values that have NULL marks to be part of the data. The number of the generated tuples depends on the type experiment. However, we ensured that the generated tuples have no duplicate tuples using a manual check and an algorithm. We derived 36 currency constraints. We use this dataset to test the scalability and the quality of our techniques.

### Parameters

The experiments were run on a Windows machine with a 2.00GHz Intel Core i7 CPU, and 8GB of RAM. We let $d_{num}$ be the number of tuples pertaining to an entity, i.e.,

[1]`http://archive.ics.uci.edu/ml/datasets.html`

[2]`http://www.generatedata.com`

the number of duplicates, and we let $d_{rate}$ be the percentage of duplicate tuples in the data, i.e., the rate of duplicates. Let $e$ be the fraction of the tuples in the duplicate tuples that are modified to have errors, i.e. the rate of error . Let $N$ be size of data $D_d$, including the duplicates.

## 6.2 Scalability Evaluation

We used the *Clients* dataset to evaluate the scalability of our data cleaning techniques w.r.t. running times. We varied the parameters of interest to evaluate the impact of change on the performance.

*Exp-1: Scalability with the number of duplicates*: Figure 6.1 depicts the running times against increasing $d_{num}$. We fix $d_{rate} = 50\%$, $N = 4000$, and $e = 12\%$. The running time decreases as $d_{num}$ increases. This is due to the fact that increasing $d_{num}$ results in a lower number of clusters in $N$ because there are more duplicates per cluster, which makes the total running time decrease. That is, having more duplicates per cluster increases the time to clean each one, but the total number of clusters is less as $N$ is fixed.

*Exp-2: Scalability with the rate of duplicates*: In this experiment, we measure the running time while varying the value of $d_{rate}$ from 30% to 70%. We fix $d_{num} = 4$, $N = 4000$, and $e = 12\%$. Figure 6.2 shows the results of this experiment. We notice that the running time moderately increases with the increase of $d_{rate}$. This is due to the fact that increasing $d_{rate}$ increases the number of clusters, which in turn, increases the total time required to clean the data. For instance, with a 50% rate of duplicates, there are 3200 duplicate tuples, while with a 70% rate of duplicates, there are 3612 duplicates. It is moderate because $N$ is fixed.

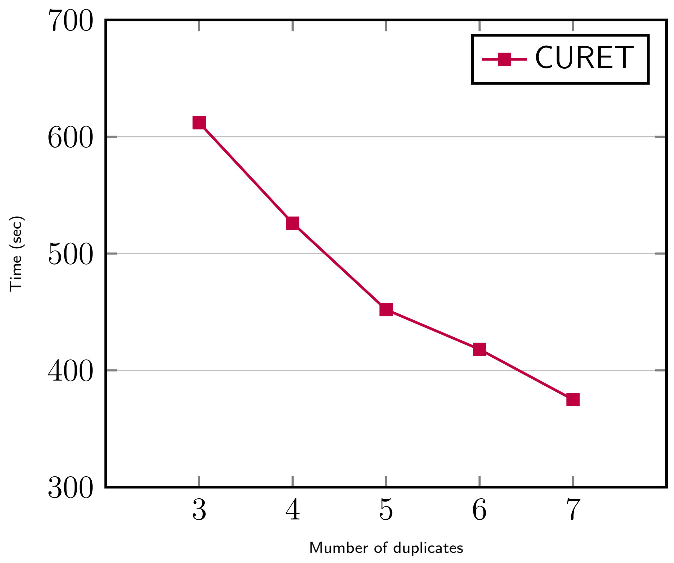

Figure 6.1: Scalability with number of duplicates

This confirms the result of the previous experiment that increasing the number of clusters has more impact on the performance than increasing the cluster size.

*Exp-3: Scalability with the number of tuples*: We study the impact of increasing the number of tuples on the running time. We fix $d_{num}$ = 4, $d_{rate}$ = 50%, and $e$ = 12%. Figure 6.3 illustrates that increasing the number of tuples does affect the running time. Comparing to figures 6.1 and 6.2, it grows more rapidly.

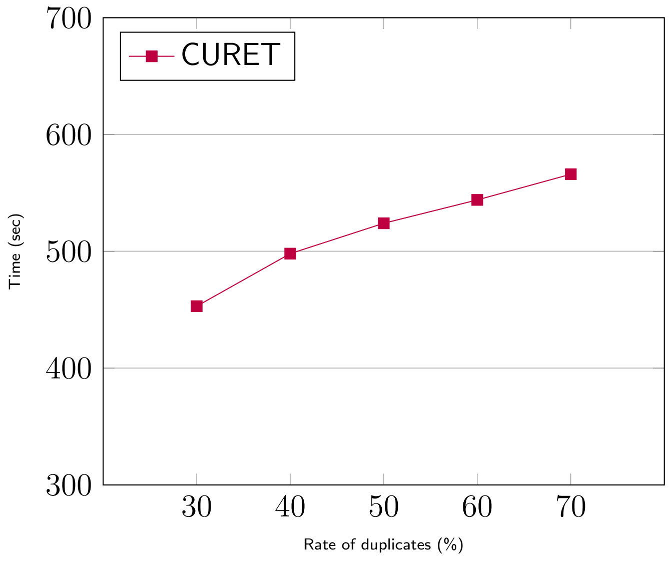

Figure 6.2: Scalability with rate of duplicates

This is due to the fact that increasing $N$ increases the number of clusters in a higher rate compared to experiment 2. For instance, for $N = 6000$, there are 4800 duplicate tuples, while for $N = 10000$, there are 80000 duplicate tuples. This makes the number of clusters increase more rapidly, in which each cluster needs time to compute the trustworthiness of its attribute values.

*Exp-4: Scalability with the rate of error*: We evaluate the effect of increasing the error rate $e$ on the performance. We fix $d_{num} = 4$, $d_{rate} = 50\%$, and $N = 4000$. Figure 6.4 shows the results of different values of $e$, including 1%, 3%, 5%, 9%, and 12%. When increasing $e$, the running time slightly increases. The shows that the impact of increasing $e$ is low compared to the impact of increasing the rate of duplicates and number of tuples in figures 6.2 and 6.3, respectively. The reason is that the running times required to process different attribute values for each cluster are close to one another for different clusters, as $d_{rate}$ and $d_{num}$ are fixed. However, having more errors adds a small amount of extra time to compute the trustworthiness.

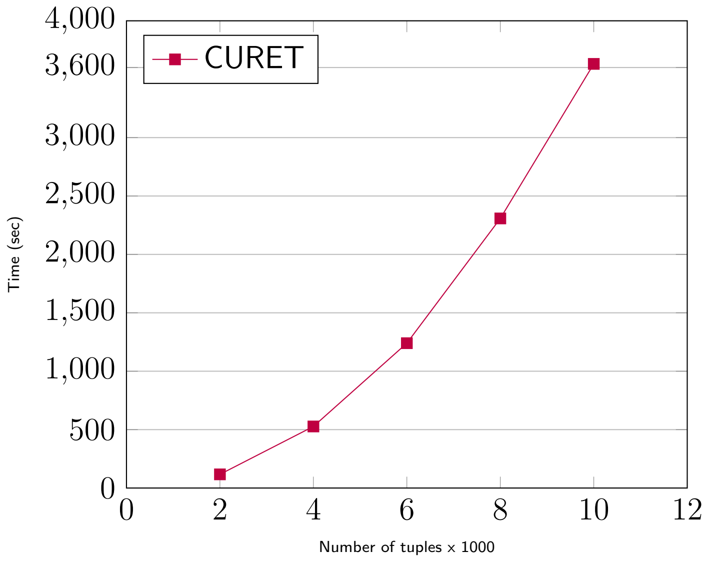

Figure 6.3: Scalability with number of tuples

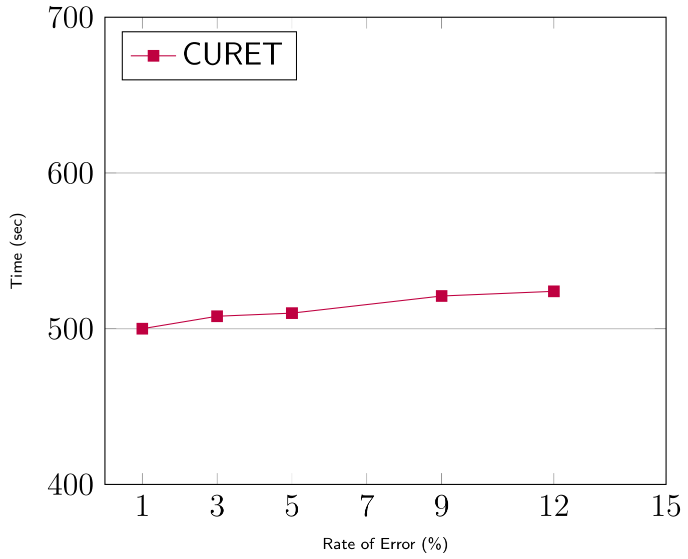

Figure 6.4: Scalability with the rate of error

## 6.3 Qualitative Evaluation

We used the F-measure to test the quality of our approach. F-measure is defined as:

$F - measure = \frac{2 \times Precision \times Recall}{Precision + recall}$

where precision is the number of correctly identified duplicates over the number of declared duplicates; and recall is the number of correctly identified duplicates over the number of the true duplicates.

*Exp-5: F-measure using Census Income dataset*: Figure 6.5 illustrates the F-measure values for the *Census Income* dataset at various rates of error. We fix $d_{num} = 4$, $d_{rate} = 25\%$, and $N = 1050$. The results show that F-measure values are as high as 0.9, 0.89, and 0.889 for $e$ rates 1%, 3%, and 5%, respectively. For $e$ rates 9% and 12%, the F-measure values are 0.831 and 0.827, respectively. According to Redman (1998), the data error rates in enterprises are approximately 1-5%. This indicates that our techniques are able to clean the data effectively for this range of error. In addition, although the F-measure rate decreases with the increase of $e$, as expected, it is still able to achieve good F-measure values for $e$ rates of 9% and 12%.

*Exp-6: F-measure using Clients dataset*: In this experiment we evaluated the quality of our techniques using *Clients* dataset. The results are shown in figure 6.6. We fix $d_{num} = 4$, $d_{rate} = 50\%$, and $N = 4000$. The values of F-measure decrease as the error rate increases, as expected. However, an F-measure value of 0.85 is achieved for a 12% error rate, which indicates that our techniques are cleaning the data effectively even with higher error rates.

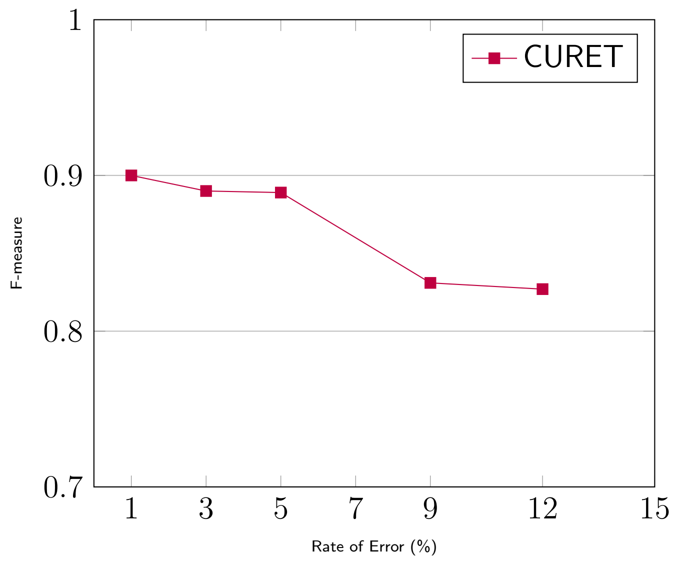

Figure 6.5: F-measure: *Census Income* dataset

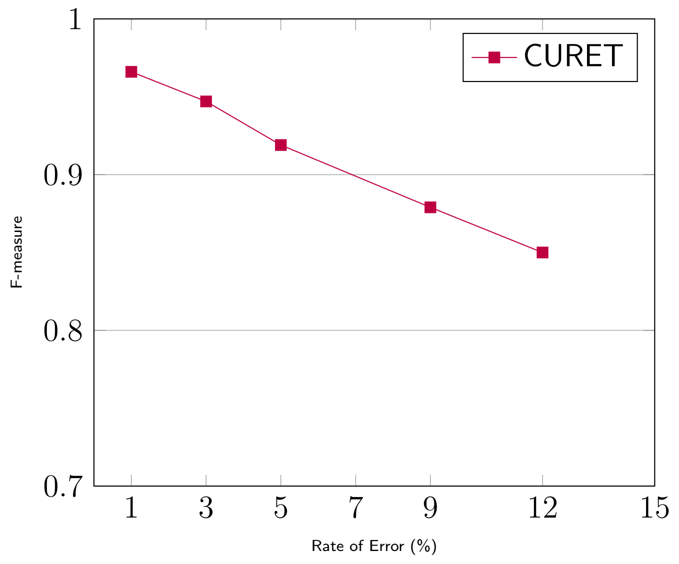

Figure 6.6: F-measure: *Clients* dataset

## 6.4 Comparative Study

A work that is close to our work is (Fan *et al.*, 2014a). As discussed earlier in section 5.7, they propose an approach for conflict resolution that combines data currency (Fan *et al.*, 2011) and data consistency (Fan *et al.*, 2008) techniques. Our approach is different because it resolves the outdated values of the duplicate tuples using currency orders and currency constraints while also using density information. In addition, it is different because it fixes the inaccuracy issues that result from issues such as transcription errors and lack of standards for the attribute values that do not necessarily have currency orders and currency constraints. Our approach doesn't resolve the inconsistency issues.

The experiments performed in (Fan *et al.*, 2014a) include evaluating their algorithm using currency constraints only, and using both currency constraints and CFDs. We implemented their algorithm, denoted Conf-Resol, to resolve conflict resolution using currency constraints without manual user interaction. It does not repair the inconsistent values. We used the *Clients* dataset and the same parameter settings of experiment 6. Figure 6.7 shows the results of the F-measure values. Our algorithm achieved better F-measure quality values than Conf-Resol. This indicates that corroborating data currency and accuracy through density information improves the quality of data cleaning, while considering multiple quality issues and reducing human effort.

The algorithm in (Fan *et al.*, 2014a) may also resolve conflict resolution by interleaving the inferences of data currency and data consistency. We compare our results using *Clients* dataset with their results using their datasets. Here, the comparison is an approximated comparison as the datasets are different in addition to some differences in the parameters settings. Since one of the main objectives of our approach is to reduce human effort, we compare our results with both results that involved and did not involve manual user interaction, for the purpose of contrasting the two techniques. Their F-measure values below are taken from (Fan *et al.*, 2014a) and have been written manually. In the results of (Fan *et al.*, 2014a) that involved manual user interaction, combining currency with consistency achieved an average F-measure value of 0.93, with a quality improvement by an average of 7.6%, over the results using only currency constraints. For their results without user interaction, combining currency with consistency achieved an average F-measure value of 0.84 with a quality improvement by an average of 5.6% over the results using only currency constraints. Our values of F-measure in figure 8 show an average of 0.91 with a quality improvement by an average of 5.7% over the results using only currency constraints, in contrast to 0.84 and 5.6% for the algorithm in (Fan *et al.*, 2014a). Moreover, our F-measure value is even closer to the value of 9.3 in (Fan *et al.*, 2014a) that involved user interaction. This shows that our algorithm improved the quality of data cleaning without manual user interaction.

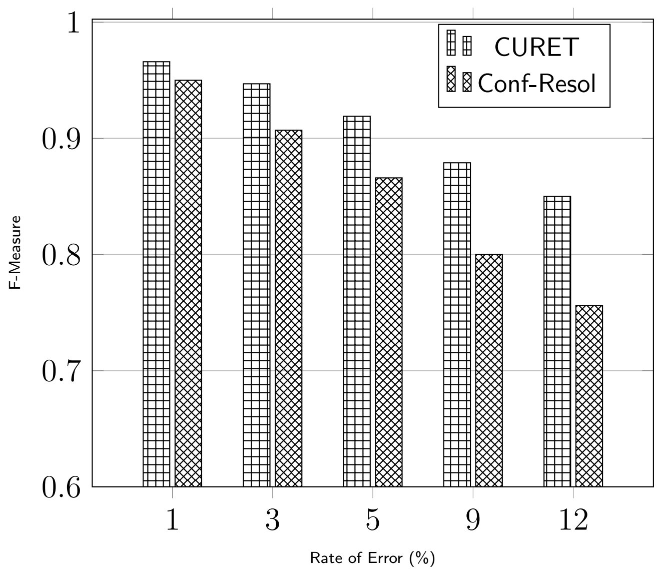

Figure 6.7: The comparison of quality of CURET algorithm compared to Conf-Resol algorithm using only $\Gamma$

# Chapter 7

# Future Work

In this thesis, we presented techniques to clean the data holistically, as opposed to solving individual types of errors separately. In this Chapter, we present several directions we envision and would like to pursue in our future work.

## 7.1 Error Type-Guided Data Cleaning

Different causes may introduce errors in data such as bad processing of optical character recognition (OCR) devices, integration of multiple data sources, transcription errors, data entry over the phone (i.e. phonetic errors), or sampling errors. The characteristics and distribution of errors would be different according to the error type. An earlier work by Damerau (1964) stated that 80% of all errors are single typographic errors that include insertion, substitution, or deletions of a single letter and transpositions of two adjacent characters. Similar results have been found by Peterson (1986). The work of Pollock and Zamora (1984) states that misspellings in raw keyboarding that contain only a single character error typically represents over

90% of keyboarding errors. They also state that single character errors happen more often in the middle or toward the end of a word. According to Kukich (1992) substitution errors between characters that look similar usually happen in optical character recognition errors (e.g., ‘t’ and ‘f’) and transcription typing errors tend to happen between adjacent keys in the keyboard.

The current focus of data cleaning systems is to fix errors without being aware of their sources. Since the current methods do not analyze the patterns of the erroneous data values, this may affect the quality of cleaning, which in turn may affect the quality of the data. For example, in repairing CFD violations, let us assume that we have data that was generated using OCR from handwritten documents. We may have the following three attribute values in three tuples: “root5”, “roofs”, “roods”. The CFD states that we have only two values, either “roofs” or “roods”. The first value is more likely to be originally “roofs” not “roods”, as the characters ‘t’ and ‘f’ are one of the possible OCR errors. Another possible error is that the end character may be either ‘s’ or ‘5’. So we conclude that there are two values of “roofs” versus one value of “roods”, which will help to guide the cleaning, so the overall data cleaning quality would be improved.

Our objective is to incorporate the functionality of error-type awareness into our data cleaning approach to propose fixes that can be effectively guided by the potential causes of the errors.

## 7.2 Extended Data Repair Model

Our data cleaning approach considers only the violations of integrity constraints of functional dependencies. Our objective is to consider in our approach other types of integrity constraints including inclusion dependencies (INDs) and conditional functional dependencies (CFDs).

Consider the inclusion dependency: $Orders[TruckID] \subseteq Truck[Number]$, introduced in subsection 2.5.3. A possible interesting repair to inconsistent data w.r.t. the IND would be adding or modifying an attribute value either to *Truck* or *Orders* through incorporating error type-guided data cleaning technique to the repair.

By using CFDs, discussed earlier in subsection 2.5.2, a repair operation may fix the erroneous values according to the constant values in the pattern tableau. These values may be incorporated into our cost model, in order to help make decisions about choosing the best repair.

## 7.3 Interleaving Cleaning and Clustering of Multiple Data Quality Issues

Clustering algorithms are used to help in the data cleaning process. Current systems and techniques usually separate clustering of tuples from data cleaning process. A recently proposed approach (Song *et al.*, 2015) suggests that data be clustered and repaired at the same time. This is based on the minimum change principle, which has been discussed previously. The goal of this work is to improve clustering by repairing and utilizing dirty data.

By developing algorithms that interleave cleaning and clustering, we envision a direction in which the goal is to improve the cleaning of multiple data quality issues. It would be interesting to incorporate the technique of (Song *et al.*, 2015) into the data cleaning approaches. For example, to clean duplicate records that have inaccurate and outdated data, we would utilize our corroborating and confidence models to improve the clustering by extending the technique to handle multiple data quality issues in the clustering. At the same time, the results of our data cleaning techniques would identify a single tuple for a set of tuples pertaining to the same entity. As such, we would end up with better quality clusters and a higher quality of data cleaning. In doing this, we also envision combining our techniques of deduplication with data consistency and accuracy for better handling of data quality issues.

## 7.4 Fixing Incomplete Data

In future work, we would like to expand our algorithms to fix other types of data quality issues including incomplete data. In a rational database, there are two kinds of completeness, closed world assumption (CWA) and open world assumption (OWA). In CWA, only the values of real-world entities actually reside in the database. In OWA, facts not represented in the tuples of the database cannot be determined to be true or false. (Batini and Scannapieca, 2006). Databases are often incomplete and both attribute values and tuples could be missing. A database is neither entirely closed-world nor entirely open-world (Fan and Geerts, 2012).

In a model with NULL values with CWA, the completeness of a tuple w.r.t. its attributes values characterizes tuple completeness. The number of missing values of a specific attribute in a table characterizes attribute completeness (Batini and Scannapieca, 2006). We envision a direction that could be used to fix incomplete data through incorporating our weighting, corroborating, and confidence models that are based on the density of data, in order to model completeness for tuples and attributes that have missing values. Confidence scores can be assigned to measure and contrast between the attribute values in the tuples or between the values of an attribute.

# Chapter 8

# Conclusions

In this thesis, we aimed to leverage fixing multiple data quality issues through utilizing data density in data cleaning, while reducing manual human effort and the dependency on master data or training data. We applied this approach to multiple data quality issues: eliminating duplicates, repairing functional dependency violations, correcting inaccurate data, and curtailing outdated data, where the erroneous values in data are introduced because of multiple causes.

We have studied a problem in data quality in which we are given unclean databases that have duplicate records, inconsistent data, and inaccurate data quality issues. We presented algorithms for fixing these issues holistically, without the reliance on manual user interaction, master data, or training data. The algorithms utilize density information to clean the data where the deduplication process is integrated with data repair and discovery of accurate values. We presented a weight model that is based on the density of data to determine the weights of tuples. The weights represent the confidence in the correctness in the values of the data. We presented a cost model that is based on the weight model for repairing a database that is inconsistent with respect to a given set of functional dependencies (FDs). To discover the accurate attribute values in the duplicate tuples, we measured the relatedness of the words of the attributes based on the hierarchical clustering. We have not used pairwise comparison techniques (Janicki and Soudkhah, 2015; Kułakowski *et al.*, 2014) in data classifications in this thesis, however, incorporating these techniques to ours would be interesting to pursue in the future.

We conducted an experimental evaluation to verify the quality and scalability of our algorithms in various configurations, in addition to conducting a comparative evaluation. We used synthetic and real datasets. Our experiments showed that our density-based data cleaning algorithms are effective in fixing the erroneous values.

We have also presented algorithms for cleaning the data that have multiple data quality issues including duplicate records, outdated data, and inaccurate data. As stated previously, these algorithms do not rely on manual user interaction, master data, or training data. We considered the problem of deduplication in which there are duplicate tuples, outdated data, and inaccurate data that are caused by transcription errors and a lack of standard formats. We introduced a corroborating model for attribute values based on a confidence model and the density of the data where the tuples in the dense regions are packed together. The confidence model of the tuples is represented as confidence scores about the correctness of the tuples with respect to the currency orders and currency constraints. We did not assume the availability of timestamps and we assumed that currency orders and currency constraints do not necessarily provide all the information required to discover the most current attribute values.

Using synthetic and real datasets, our experiments showed that our algorithms are effectively able to address these discrepancies in data by utilizing the density information. We presented comparative evaluation that revealed that our approach was able to identify a single tuple for each set of duplicate tuples referring to the same real-world entity.

# Bibliography

Abiteboul, S., Hull, R., and Vianu, V. (1995). *Foundations of Databases*. Addison-Wesley.

Al-janabi, S. and Janicki, R. (2016a). Corroborating quality of data through density information. In *IntelliSys* (Accepted for publication).

Al-janabi, S. and Janicki, R. (2016b). A density-based data cleaning approach for deduplication with data consistency and accuracy. In *SAI* (Accepted for publication).

Baker, L. and McCallum, A. (1998). Distributional clustering of words for text classification. In *SIGIR*, pages 96–103.

Batini, C. and Scannapieca, M. (2006). *Data Quality Concepts, Methodologies and Techniques*. Springer-Verlag.

Beskales, G., Ilyas, I., and Golab, L. (2010). Sampling the repairs of functional dependency violations under hard constraints. *VLDB Endowment*, **3**(1-2), 197–207.

Bleiholder, J. and Naumann, F. (2009). Data fusion. *CSUR*, **41**(1), Article No. 1.

Bohannon, P., Fan, W., Flaster, M., and Rastogi, R. (2005). A cost-based model and effective heuristic for repairing constraints by value modification. In *SIGMOD*, pages 143–154.

Bohannon, P., Fan, W., Geerts, F., Jia, X., and Kementsietsidis, A. (2007). Conditional functional dependencies for data cleaning. In *ICDE*.

Bravo, L., Fan, W., and Ma, S. (2007). Extending dependencies with conditions. In *VLDB*.

Cao, Y., Fan, W., and Yu, W. (2013). Determining the relative accuracy of attributes. In *SIGMOD*, pages 565–576.

Chaudhuri, S., Ganti, V., and Motwan, R. (2005). Robust identification of fuzzy duplicates. In *ICDE*, pages 865–876.

Chen, M., Han, J., and Yu, P. (1996). Data mining: an overview from a database perspective. *Trans. Knowl. Data Eng.*, **8**(6), 866–883.

Cheng, Y. (1995). Mean shift, mode seeking, and clustering. *IEEE Trans. Pattern Anal. Mach. Intell.*, **17**(8), 790799.

Chomicki, J. and Marcinkowski, J. (2005). Minimal-change integrity maintenance using tuple deletions. *Information and Computation*, **197**(1-2), 90–121.

Chomicki, J. and Toman, D. (2005). Temporal databases. In *Handbook of Temporal Reasoning in Artificial Intelligence*. Elsevier.

Cios, K., Pedrycz, W., and Swiniarski, R. (1998). *Data Mining Methods for Knowledge Discovery*. Kluwer Academic Publishers.

Cohen, W. and Richman, J. (2002). Learning to match and cluster large high-dimensional data sets for data integration. In *SIGKDD*.

Comaniciu, D. and Meer, P. (1999). Mean shift analysis and applications. In *IEEE International Conference on Computer Vision*.

Comaniciu, D. and Meer, P. (2002). Mean shift: a robust approach toward feature space analysis. *IEEE Trans. Pattern Anal. Mach. Intell.*, **24**(5), 603–619.

Cong, G., Fan, W., Geerts, F., Jia, X., and Ma, S. (2007). Improving data quality: Consistency and accuracy. In *VLDB*, pages 315–326.

Dallachiesa, M., Ebaid, A., Eldawy, A., Elmagarmid, A., Ilyas, I., Ouzzani, M., and Tang, N. (2013). NADEEF: A commodity data cleaning system. In *SIGMOD*, pages 541–552.

Damerau, F. (1964). A technique for computer detection and correction of spelling errors. *Communications of the ACM*, **7**(3), 171176.

Dasu, T. and Johnson, T. (2003). *Exploratory Data Mining and Data Cleaning*. Wiley-Interscience.

Dhillon, I., Mallela, S., and Kumar, R. (2002). Enhanced word clustering for hierarchical text classification. In *SIGKDD*, pages 191–200.

Dong, X., Berti-Equille, L., and Srivastava, D. (2009). Truth discovery and copying detection in a dynamic world. *VLDB Endowment*, **2**(1), 562–573.

Eckerson, W. (2002). Data quality and the bottom line: Achieving business success through a commitment to high quality data. In *The Data Warehousing Institute*.

Elmagarmid, A., Ipeirotis, P., and Verykios, V. (2007). Duplicate record detection: A survey. *Trans. Knowl. Data Eng.*, **19**(1), 1–16.

Ester, M., Kriegel, H., Sander, J., and Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, pages 226–231.

Fan, W. (2008). Dependencies revisited for improving data quality. In *PODS*.

Fan, W. and Geerts, F. (2010). Capturing missing tuples and missing values. In *PODS*.

Fan, W. and Geerts, F. (2012). *Foundations of Data Quality Management*. Morgan & Claypool Publishers.

Fan, W., Geerts, F., Jia, X., and Kementsietsidis, A. (2008). Conditional functional dependencies for capturing data inconsistencies. *TODS*, **33**(2).

Fan, W., Geerts, F., and Wijsen, J. (2011). Determining the currency of data. In *PODS*.

Fan, W., Li, J., Ma, S., and Yu, N. T. W. (2012). Towards certain fixes with editing rules and master data. *VLDB*, **21**(2), 213–238.

Fan, W., Geerts, F., Tang, N., and Yu, W. (2014a). Conflict resolution with data currency and consistency. *JDIQ*, **5**(1-2, Article No. 6).

Fan, W., Ma, S., Tang, N., and Yu, W. (2014b). Interaction between record matching and data repairing. *JDIQ*, **4**(4), Article 16.

Fukunaga, K. and Hostetler, L. (1975). The estimation of the gradient of a density function, with applications in pattern- recognition. *IEEE Trans. Inf. Theory*, **21**(1), 32–40.

Galhardas, H., Florescu, D., Simon, D. S. E., and Saita, C.-A. (2001). Declarative data cleaning: Language, model, and algorithms. In *VLDB*, pages 371–380.

Hernández, M. and Stolfo, S. (1998). Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery*, **2**(1), 9–37.

Herzog, T., Scheuren, F., and Winkler, W. (2007). *Data Quality and Record Linkage Techniques*. Springer.

Jaccard, P. (1901). Étude comparative de la distribution florale dans une portion des Alpes et des Jura. *Bulletin de la Société Vaudoise des Sciences Naturalles*, **37**, 547–579.

Jain, A. and Dubes, R. (1988). *Algorithms for Clustering Data.* Prentice-Hall Inc.

Jain, A. K., Murty, M. N., and Flynn, P. J. (1999). Data clustering: A review. *CSUR*, **31**(3), 264–323.

Janicki, R. and Lenarcic, A. (2016). Optimal approximations with rough sets and similarities in measure spaces. *International Journal of Approximate Reasoning*, **71**, 1–14.

Janicki, R. and Soudkhah, M. H. (2015). On classification with pairwise comparisons, support vector machines and feature domain overlapping. *The Computer Journal*, **58**(3), 416–431.

Januzaj, E., Kriegel, H.-P., and Pfeifle, M. (2003). Towards effective and efficient distributed clustering. In *ICDM*, pages 49–58.

Kaufman, L. and Rousseeuw, P. (2005). *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley.

Kolahi, S. and Lakshmanan, L. (2009). On approximating optimum repairs for functional dependency violations. In *ICDT*, pages 53–62.

Kukich, K. (1992). Techniques for automatically correcting words in text. *ACM computing surveys*, **24**(4), 377439.

Kułakowski, K., Szybowski, J., and Tadeusiewicz, R. (2014). Tender with success - the pairwise comparisons approach. *In Knowledge-Based and Intelligent Information & Engineering Systems 18th Annual Conference, KES-2014, Gdynia, Poland*, **35**, 1122–1131.

Larsen, B. and Aone, C. (1999). Fast and effective text mining using linear-time document clustering. In *KDD*.

Le, M., Ling, T., and Low, W. (2000). Intelliclean: a knowledge-based intelligent data cleaner. In *SIGKDD*, pages 290–294.

Lee, Y., Pipino, L., Funk, J., and Wang, R. (2006). *Journey to Data Quality*. The MIT Press.

Levenshtein, V. (1965). Binary codes capable of correcting deletions, insertions and reversals. *Doklady Akademii Nauk SSSR*, **163**(4), 845–848.

Li, P., Dong, X. L., Maurino, A., and Srivastava, D. (2012). Linking temporal records. *Front. Comput. Sci.*, **6**(3).

Loshin, D. (2010). *The practitioner's Guide to Data Quality Improvement*. Morgan Kaufmann.

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. *5th Berkeley Symp on Math. Stat. and Prob.*, **1**, 281297.

Marczewski, E. and Steinhaus, H. (1958). On a certain distance of sets and corresponding distance of functions. *Colloquium Mathematicum*, **6**(1), 319–327.

Marsh, R. (2005). Drowning in dirty data? it's time to sink or swim: A four-stage methodology for total data quality management. *Database Marketing* & *Customer Strategy Management*, **12**(2), 105–112.

McCallum, A. and Wellner, B. (2004). Conditional models of identity uncertainty with application to noun coreference. In *NIPS*.

McCallum, A., Nigam, K., and Ungar, L. (2000). Efficient clustering of high-dimensional data sets with application to reference matching. In *KDD*.

Monge, A. and Elkan, C. (1997). An efficient domain-independent algorithm for detecting approximately duplicate database records. In *SIGMOD*.

Müller, H. and Freytag, J. (2003). Problems, methods, and challenges in comprehensive data cleansing. In *Humboldt-Universität zu Berlin zu Berlin*.

Mulry, M., Bean, S., Bauder, M., Wagner, D., Mule, T., and Petroni, R. (2006). Evaluation of estimates of census duplication using administrative records information. *Journal of Official Statistics*, **22**(4), 655–679.

Naumann, F. and Herschel, M. (2010). *An Introduction to Duplicate Detection*. Morgan & Claypool Publishers.

Peterson, J. (1986). A note on undetected typing errors. *Communications of the ACM*, **29**(7), 633637.

Pollock, J. and Zamora, A. (1984). Automatic spelling correction in scientific and scholarly text. *Communications of the ACM*, **27**(4), 358–368.

Raman, V. and Hellerstein, J. (2001). Potter's wheel: An interactive data cleaning system. In *VLDB*, pages 381–390.

Ravikumar, P. and Cohen, W. (2004). A hierarchical graphical model for record linkage. In *UAI*.

Redman, T. (1996). *Data Quality for the Information Age*. Artech House.

Redman, T. (1998). The impact of poor data quality on the typical enterprise. *Communications of the ACM*, **41**(2), 79–82.

Sarawagi, S. and Bhamidipaty, A. (2002). Interactive deduplication using active learning. In *SIGKDD*.

Slonim, N. and Tishby, N. (2001). The power of word clusters for text classification. In *ECIR*.

Song, S., Li, C., and Zhang, X. (2015). Turn waste into wealth: On simultaneous clustering and cleaning over dirty data. In *SIGKDD*.

Tejada, S., Knoblock, C., and Minton, S. (2002). Learning domain-independent string transformation weights for high accuracy object identification. In *SIGKDD*.

Ukkonen, E. (1992). Approximate string-matching with q-grams and maximal matches. *Theoretical Computer Science*, **92**(1), 191–211.

Verykios, V., Elmagarmid, A., and Houstis, E. (2000). Automating the approximate record-matching process. *Information Sciences*, **126**(1-4), 83–98.

Watts, S., Shankaranarayanan, G., and Even, A. (2009). Data quality assessment in context: A cognitive perspective. *Decision Support Systems*, **48**(1), 202–211.

Winkler, W. (1999). The state of record linkage and current research problems. Technical report, US Bureau of the Census.

Winkler, W. (2004). Methods for evaluating and creating data quality. *Information Systems*, **29**(7), 531–550.

Winkler, W. (2006). Overview of record linkage and current research directions. Technical report, US Bureau of the Census.

Witten, I., Frank, E., and Hall, M. (2011). *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann.

Wu, M. and Marian, A. (2011). A framework for corroborating answers from multiple web sources. *Information Systems*, **36**(2), 431–449.

Xu, R. and Wunsch, D. (2005). Survey of clustering algorithms. *IEEE Trans. Neural Networks*, **16**(3), 645–678.

Yakout, M., Elmagarmid, A., Neville, J., Ouzzani, M., and Ilyas, I. (2011). Guided data repair. *VLDB Endowment*, **4**(5), 279–289.

Yang, Y. and Pedersen, J. (1997). A comparative study on feature selection in text categorization. In *ICML*, pages 412–420.

Yin, X., Han, J., and Yu, P. (2008). Truth discovery with multiple conflicting information providers on the web. *Trans. Knowl. Data Eng.*, **20**(6), 796–808.

Zhang, H., Diao, Y., and Immerman, N. (2010). Recognizing patterns in streams with imprecise timestamps. *VLDB*, **3**(1-2).