Data Quality and Data Cleaning

in

Database Applications

Lin Li

A thesis submitted in partial fulfilment of
the requirements of Edinburgh Napier University
for the award of
Doctor of Philosophy

School of Computing

September 2012

ABSTRACT

Today, data plays an important role in people's daily activities. With the help of some database applications such as decision support systems and customer relationship management systems (CRM), useful information or knowledge could be derived from large quantities of data. However, investigations show that many such applications fail to work successfully. There are many reasons to cause the failure, such as poor system infrastructure design or query performance. But nothing is more certain to yield failure than lack of concern for the issue of data quality. High quality of data is a key to today's business success. The quality of any large real world data set depends on a number of factors among which the source of the data is often the crucial factor. It has now been recognized that an inordinate proportion of data in most data sources is dirty. Obviously, a database application with a high proportion of dirty data is not reliable for the purpose of data mining or deriving business intelligence and the quality of decisions made on the basis of such business intelligence is also unreliable. In order to ensure high quality of data, enterprises need to have a process, methodologies and resources to monitor and analyze the quality of data, methodologies for preventing and/or detecting and repairing dirty data. This thesis is focusing on the improvement of data quality in database applications with the help of current data cleaning methods. It provides a systematic and comparative description of the research issues related to the improvement of the quality of data, and has addressed a number of research issues related to data cleaning.

In the first part of the thesis, related literature of data cleaning and data quality are reviewed and discussed. Building on this research, a rule-based taxonomy of dirty data is proposed in the second part of the thesis. The proposed taxonomy not only summarizes the most dirty data types but is the basis on which the proposed method for solving the Dirty Data Selection (DDS) problem during the data cleaning process

was developed. This helps us to design the DDS process in the proposed data cleaning framework described in the third part of the thesis. This framework retains the most appealing characteristics of existing data cleaning approaches, and improves the efficiency and effectiveness of data cleaning as well as the degree of automation during the data cleaning process.

Finally, a set of approximate string matching algorithms are studied and experimental work has been undertaken. Approximate string matching is an important part in many data cleaning approaches which has been well studied for many years. The experimental work in the thesis confirmed the statement that there is no clear best technique. It shows that the characteristics of data such as the size of a dataset, the error rate in a dataset, the type of strings in a dataset and even the type of typo in a string will have significant effect on the performance of the selected techniques. In addition, the characteristics of data also have effect on the selection of suitable threshold values for the selected matching algorithms. The achievements based on these experimental results provide the fundamental improvement in the design of ‘algorithm selection mechanism’ in the data cleaning framework, which enhances the performance of data cleaning system in database applications.

ACKNOWLEDGEMENT

I would like to thank my supervisors Dr. Taoxin Peng, Professor Jessie Kennedy, and my PhD panel chairs for all their help, support, expertise and understanding throughout my period of PhD study. I would also like to thank all staff at the School of Computing at Napier University, especially the members at the Centre for Information and Software Systems group, for providing me valuable feedback and suggestions during my PhD study.

Finally, great appreciation and thanks to my mum and my dad for their consistent spiritual support. I am proud of them and appreciate what they contribute to my life.

PUBLICATIONS FROM THE PHD WORK

[1] Li, L., Peng, T., & Kennedy, J. (2010). Improving Data Quality in Data Warehousing Applications. *Proceedings of the 12th International Conference on Enterprise Information Systems*, Funchal, Madeira Portugal.

[2] Li, L., Peng, T., & Kennedy, J. (2011). A Rule Based Taxonomy of Dirty Data. *GSTF International Journal on Computing*, *1*(2), 140-148.

[3] Peng, T., Li, L., & Kennedy, J. (2011). An Evaluation of Name Matching Techniques. *Proceedings of 2nd Annual International Conference on Business Intelligence and Data Warehousing*, Singapore.

[4] Peng, T., Li, L., & Kennedy, J. (2012). A Comparison of Techniques for Name Matching. *International Journal on Computing*, *2*(1), 55-61.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 INTRODUCTION

Today, data plays an important role in people's daily activities. With the help of database applications such as decision support systems and customer relationship management systems (CRM), useful information or knowledge can be derived from large quantities of data. However, investigations show that many such applications fail to work successfully. There are many reasons to cause the failure, such as poor system infrastructure design or query performance, but nothing is more certain to yield failure than lack of concern for the issue of data quality [1].

For example, from Price Waterhouse Coopers' survey in New York in 2001, 75% of 599 companies had economic losses because of data quality problems. Because their businesses are all dependent on data-driven systems such as customer relationship management and supply chain management systems, the issue remains that only 37% of the companies were "very confident" in the quality of their own data, and only 15% were "very confident" in the quality of the data of their trading partners [2].

There is a growing awareness that high quality of data is key to today's business success. The quality of any large real world data set depends on a number of factors [3-5], among which the source of the data is often the crucial factor. It has now been recognized that an inordinate proportion of data in most data sources is dirty [6]. For example, some investigations show that errors in a large data set are common and are typically around 5% unless extreme measures have been taken [7, 8]. Due to the 'garbage in, garbage out' principle, dirty data will distort information obtained from it [9]. Obviously, a database application such as a data warehouse with a high proportion of dirty data is not reliable for the purpose of data mining or deriving business intelligence and the quality of decisions made on the basis of such business intelligence is also not reliable.

Therefore, before using such databases, dirty data from them should be cleaned. That is, to ensure high quality of data, enterprises need to have a process, methodologies and resources to monitor and analyze the quality of data, and methodologies for preventing or detecting and subsequently repairing dirty data.

This thesis provides a systematic and comparative description of the research related to the improvement of the quality of data, and has addressed a number of research issues related to data cleaning. In the following sections, we briefly introduce fundamental concepts and research issues related to data cleaning and data quality.

1.1 Data Quality

Investigations into the problems related to data quality can be traced back to as early as late 1960s when a mathematical theory for considering the duplicate problem in statistical data sets was proposed by Fellegi and Sunter [10]. However, it is only in the 1990s that the data quality problem has been considered in computer science with the data stored in databases and data warehouse systems. More and more people have become aware that poor data quality is one of the main reasons for the failure of a database project. Though a variety of definitions for data quality have been given [3，8，11], studies show that still no formal definition for data quality exists [8]. From the literature, data quality can be defined as “fitness for use”, i.e., the ability of data to meet the user's requirement. The nature of this definition directly implies that the concept of data quality is relative. Orr states “the problem of data quality is fundamentally intertwined in how our system fits into the real world; in other words, with how users actually use the data in the system” [8]. This has two interpretations: one is that if a data set is available and is as good as it can be, there are no other options than to use it. The other one is that what is considered as quality data in one case may not be sufficient in other cases. For example, an analysis of the financial position of a company may require data in units of thousands of pounds

while an auditor requires precision to the pence, i.e. in real life, it is the business policy or business rules that determine whether or not the data is of quality.

Generally speaking, data quality can be measured or assessed with a set of characteristics or quality properties called data quality dimensions [4]. Some commonly used data quality dimensions include accuracy, completeness, timeliness, and consistency, which can be refined as:

- Accuracy – conformity of the recorded value with the actual value;
- Timeliness – the recorded value is not out of date;
- Completeness – all values for a certain variable are recorded;
- Consistency – the representation of data is uniform in all cases.

Therefore, data quality can be considered as a multi-dimensional concept. These data quality dimensions measure data quality from different angles. Within each of these dimensions, a set of data quality rules generated by real business polices can be used to make an assessment of the data quality reflected by each dimension [12]. For example, a data quality rule defined as ‘the value of date must follow the pattern of DD/MM/YYYY’ can be used for the consistency dimension. These data quality dimensions as well as data quality rules will be reviewed in detail in Chapter 2 and Chapter 3 respectively.

1.2 Data Cleaning

There is no commonly agreed formal definition of data cleaning. Depending on the particular area in which data cleaning has been applied, various definitions have been given. The major areas that include data cleaning as part of their defining processes are data warehousing, knowledge discovery in databases (KDD) and total data/information quality management (TDQM).

Within the data warehousing field, data cleaning is typically employed when several databases are merged. Records referring to the same entity are often represented in different formats in different data sets. Thus, duplicate records will appear in the merged database. The issue is to identify and eliminate these duplicates. The problem is known as the merge/purge problem [13]. Other instances of this problem are also referred to as record linkage, semantic integration, instance identification or the object identify problem in the literature [14]. There are a variety of methods proposed to address this issue: knowledge bases [9], regular expression matches and user-defined constraints [15], filtering [16], and others [17-19].

In the KDD process, data cleaning is regarded as a first step or a pre-processing step. However, no precise definition and perspective over the data cleaning process is given and data cleaning activities are performed in a very domain specific fashion. For example, Simoudis *et al* [20] defined data cleaning as the process that implements computerized methods of examining databases, detecting missing and incorrect data, and correcting errors. In data mining, data cleaning is emphasized with respect to the garbage in garbage out principle and its own techniques such as outlier detection where the goal is to find exceptions For example, the problem of outlier detection where the goal is to find exceptions [21, 22] can be used in data cleaning.

Total data quality management is an area of interest both within the research and business communities. From the literature, the data quality issue and its integration in the business process are tackled from various points of views [4, 7, 8, 23-26]. It is also referred to as the enterprise data quality management problem. However, none of the literature refers to the data cleaning problem explicitly. Most of this work deals with the process management issues from the data quality perspective, others with the definition of data quality. Of particular interest in this area, the definition of

data quality can help to define the data cleaning process to some extent. For example, within the model of data life cycles proposed by Levitin and Redman [24], data acquisition and data usage cycles contain the following series of activities: assessment, analysis, adjustment, and discarding of data. This series of activities proposed in Levitin and Redman's model define the data cleaning process from the perspective of data quality. Fox *et al* [23] proposed four data quality dimensions of the data, i.e., accuracy, currentness, completeness and consistency. The correctness of data is defined in terms of these dimensions. Thus, the data cleaning process within Fox *et al*'s data quality framework can be defined as the process that assesses the correctness of data and improves its quality.

With the above in mind and related literature [27, 28], data cleaning must be viewed as a process which is tied directly to data acquisition and definition or is applied to improving data quality in an existing system. For example, in Müller and Freytag's work, comprehensive data cleaning is defined as the entirety of operations performed on existing data to remove anomalies and receive a data collection being an accurate and unique representation of the mini-world [27]. According to Müller and Freytag's work, the three major steps within the data cleaning process are (i) define and determine error types, (ii) search and identify error instances, and (iii) correct the uncovered errors. Müller and Freytag include four major steps within the process of data cleaning: (i) auditing data to identify the types of anomalies reducing the data quality, (ii) choosing appropriate methods to automatically detect and remove them (specification of data cleaning), (iii) applying the methods to the tuples in the data collection (execution of data cleaning), and (iv) the post-processing or control step where the results are checked and the exception handling for tuples not corrected within the actual processing are handled.

The following figure (Fig.1.1) demonstrates these four major steps in the data cleaning process.

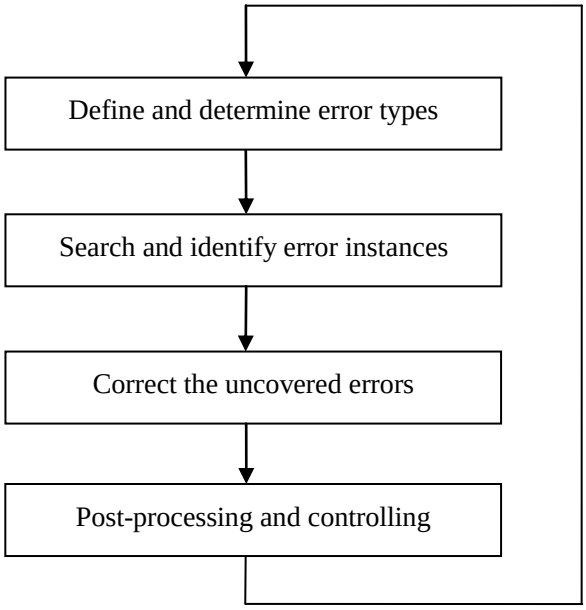


Fig.1.1 A data cleaning process

Each of these phases constitutes a set of complex problems, and a wide variety of specialized methods and technologies can be associated and applied during each phase. In this thesis, the main focus here is on the first two aspects, i.e., define and determine error types, search and identify error instances. The later aspects are very difficult to automate outside of a strict and well defined domain [21].

(i) Define and determine error types

Research shows that many enterprises do not pay adequate attention to the existence of dirty data and have not applied useful methodologies to ensure high quality data for their applications. One of the reasons is a lack of appreciation of the types and extent of dirty data [29]. Therefore, in order to improve data quality, it is necessary to understand the wide variety of dirty data that may exist within the data source as well as how to deal with them. This step is trying to discover the possible dirty data types that may exist among different data sources. From the literature, some work has been undertaken exclusively to identify problems (dirty data types) that affect

data quality and has resulted in different taxonomies of dirty data [6, 27, 28, 30]. These works are reviewed in detail in Chapter 2.

Data cleaning is a labour-intensive, time-consuming and expensive process. In practice, cleaning all dirty data types introduced by Oliveira *et al* or Kim *et al* is unrealistic and simply not cost-effective when taking into account the specific needs of a business enterprise [6, 30]. Although some research has proposed a large collection of dirty data types, such as a collection of 35 dirty data types by Oliveira *et al*, by only looking at these dirty data types it is difficult to tell which group of dirty data should be considered when facing a special requirement from a business enterprise and it would be very expensive for the system to run all algorithms for all the possible dirty data candidates. This problem is defined as the Dirty Data Selection (DDS) problem in this thesis.

In this thesis, a novel rule-based taxonomy of dirty data is proposed. Compared with existing work [6, 30], this taxonomy provides a larger collection of dirty data types than any of existing taxonomies. With the help of the proposed taxonomy, a new classification of dirty data based on data quality dimensions is proposed. It can be used by business enterprises to solve the DDS problem by prioritizing the expensive process of data cleaning, therefore maximally benefitting their organizations. This rule-based taxonomy of dirty data will be introduced in Chapter 3.

(ii) Search and identify error instances

Before the execution of this step, information regarding the dirty data types identified within the data sources should be available, since performing data cleaning in very large databases is costly and time consuming. For each of these dirty data types, searching and identifying dirty data instances are performed with the help of an appropriate data cleaning method or algorithm which not only can

help with reducing the data cleaning time but also maximizing the degree of automation.

Choosing a proper data cleaning method is proved to be a difficult task [31]. Especially when making selection of a data cleaning method out of many alternatives. It depends on several factors such as the problem domain and, the nature of errors. Additionally, organizing the multiple data cleaning methods involved during the data cleaning process is also a difficult task [32]. The challenge here is how to improve the efficiency/effectiveness when performing data cleaning tasks (i.e., reduce the data cleaning time and improve the accuracy of the cleaning results) and how to improve the degree of automation during the data cleaning process.

From the literature, many data cleaning approaches or frameworks are developed to facilitate data cleaning. However, studying these approaches reveals that they are designed mainly for solving specific data cleaning activities such as data transformations or duplicate record detection exclusively. A general data cleaning approach that can deal with the dirty data types proposed in those existing dirty data taxonomies do not exist. Additionally, regarding the selection of a suitable data cleaning technique to deal with a specific dirty data type, either a user is required to specialize a method or a fixed method is applied to all situations in those data cleaning approaches. This, as will be shown later, not only increases the data cleaning time but also affect the effectiveness of data cleaning. These data cleaning approaches will be firstly comparatively reviewed in chapter 2 and a novel data cleaning framework will be proposed in chapter 4 with two exclusive mechanisms addressed in the proposed framework to improve the efficiency and effectiveness during the data cleaning process.

1.3 Objectives of the research

The objectives of the research are as follows:

(1) To develop a taxonomy of dirty data.

Due to the lack of appreciation of the types and extent of dirty data, the existence of dirty data within database applications may not be paid adequate attention by many enterprises. This will finally become one of the important factors to cause poor data quality in these database applications. In order to improve the data quality, it is necessary to understand the wide variety of dirty data that may exist in the data sources as well as how to deal with them. Although from the literature, some work has been done exclusively for the purpose of generating the taxonomies of dirty data [6, 30], in practice cleaning all dirty data types introduced by these taxonomies is unrealistic and not cost-effective when taking into account the needs of a business enterprise. For example, according to the taxonomy of data quality problems proposed by Oliveira *et al* [30], 35 dirty data types are presented which is considered as the most comprehensive taxonomy so far from the literature. In this case, by only showing these 35 dirty data types, it is difficult to tell which possible dirty data types should be selected to deal with for different datasets when special business needs are involved. Thus one motivation of this research is to develop a taxonomy that not only addresses as many dirty data types as possible but can also help with solving the DDS problem.

(2) To develop a novel data cleaning framework

In order to ensure the data from an organization is of high quality, cleaning dirty data existing in the different data sources in a proper way is necessary. A process which can monitor, analyze and maintain the quality of data is highly recommended. From the literature, many data cleaning approaches exist to facilitate a data cleaning

process and they are crucial to make those data cleaning techniques and methodologies involved during the data cleaning process effective. However, research studies reveal they are designed to exclusively focus on the cleaning of some specific dirty data types such as duplicate record detection or data value transformation. According to the knowledge of the author, there is no such a data cleaning tool developed with the purpose of dealing with all dirty data types mentioned in those existing dirty data taxonomies.

The ability of the selection of different groups of dirty data types to deal with under the different specific needs of a business is thus highly expected for a data cleaning approach. Besides, in those current data cleaning approaches, organizing multiple cleaning tasks in a proper cleaning sequence and selecting a suitable technique for a special data cleaning task totally depends on a user's preference in most cases. Regarding the organization of the multiple cleaning tasks, ideally, the process of detecting and correcting the dirty data should be performed automatically. However, it is known that fully automatically performing data cleaning is nearly impossible in most of cases especially when an exception happens during the cleaning process and an expert is required to make a judgement. Therefore, a semi-automatic data cleaning approach with the power of automatically organizing and ordering the associated data cleaning tasks is a challenge [27].

Regarding the selection of a proper technique for a specific data cleaning task, it is necessary that a tool should include various appropriate data cleaning methods or algorithms to deal with a specific dirty data type to cope with different problem domains. Choosing a data cleaning method or algorithm from a set of alternatives has proved to be a difficult task. It depends on several factors, such as the problem domain and the nature of errors. Currently, in the existing data cleaning approaches, algorithm selection as well as its parameter's setting depends on a user's preference in most cases. For users who have not enough knowledge and experience, an

inappropriate selection of algorithms will generate a bad cleaning result. Therefore, another challenge for a data cleaning approach is that it should not only include enough techniques for a user to choose for different problem domains but also can intelligently help the user with making a choice when it is necessary.

(3) To evaluate a set of data cleaning algorithms

As mentioned above, the selection of a suitable data cleaning method is a difficult task especially when many alternative methods are available to choose from. How to make a selection to maximize the effectiveness/efficiency of data cleaning is a challenge. Many factors are required to consider during the selection of a proper data cleaning method such as the problem domains and the nature of errors. For example, matching names is one of the important steps during the data cleaning process to deal with duplicate record detection problem. There are a number of name matching techniques available. Unfortunately, there is no existing name matching technique that performs the best in all situations. Different techniques perform differently in different situations. Therefore, a problem that every researcher or a practitioner has to face is how to select an appropriate technique for a given dataset. This problem is also mentioned in the design of the proposed data cleaning framework in Chapter 4 as how to select the appropriate algorithm for a dirty data type during the data cleaning process. An objective of this research is thus to analyze and evaluate a set of name matching algorithms and present some suggestions based on the experimental results, which can be used as guidance for researchers and practitioners to select an appropriate name matching technique in a given dataset.

1.4 Contributions to knowledge

The contributions to knowledge presented in this thesis arise from the following achievements.

(1) A rule based taxonomy of dirty data

In this thesis, a rule-based taxonomy of dirty data is proposed. A taxonomy of dirty data not only provides a framework for understanding the origins of a complete spectrum of dirty data and the impact of dirty data on database applications but also sheds light on techniques for dealing with dirty data and define a metric for measuring data quality and will provide a valuable guideline for further research and enhancement of commercial products [6].

Compared with existing work, the proposed taxonomy provides a larger collection of dirty data types (38 dirty data types) than any of the existing taxonomies. Particularly, with the help of the taxonomy, a novel data cleaning method is also proposed which can be used by business enterprises to solve the proposed DDS problem, by prioritizing the expensive process of data cleaning.

(2) A novel data cleaning framework

Data cleaning is a labour-intensive, time-consuming, and expensive process, especially when huge volumes of data are involved during the data cleaning process. In this thesis, a novel data cleaning framework has been proposed, which aims to challenge the following issues: (i) minimising the data cleaning time and improving the degree of automation in data cleaning and (ii) improving the effectiveness of data cleaning. The improvement in the efficiency/effectiveness of data cleaning and the degree of automation is realized by introducing the two unique mechanisms namely ‘algorithm ordering mechanism’ and ‘algorithm selection mechanism’ during the data cleaning process. In addition, the DDS process exclusively addressed in the proposed framework can help a business to take into account the special needs according to different businesses priority policies. This framework retains the most

appealing characteristics of existing data cleaning approaches, and enjoys being able to improve the efficiency of data cleaning in data warehouse applications.

(3) A set of recommendations for the selection of a suitable name matching algorithm

The research work includes a comprehensive comparison of five popular name matching techniques based on a series of carefully designed experimental work on different last name datasets and first name datasets. The comparison results confirmed the statement that there is no clear best technique. The size of datasets, the error rate in datasets, the type of strings in a dataset and the type of typo in a string all will have significant effects on performance of the selected techniques. The timing cost of these techniques on different datasets has also been analyzed and compared. Based on the experimental results achieved, it is suggested that the selection of a technique should depend on the nature of the datasets. A set of recommendations as well as all related experimental results are presented in this thesis to help with the selection of a suitable name matching algorithm for a specific dataset.

1.5 The structure of the thesis

Chapter 1 introduces the research and the problem statement, the aim and objectives of the research are then discussed, and the contributions to knowledge are introduced.

The literature review is presented in chapter 2. Research exclusively related to dirty data type classification/taxonomy from the literature are firstly reviewed and discussed. Data cleaning methods and approaches are studied and compared in detail secondly. Finally, the literature regarding data quality and data quality dimensions

are reviewed and compared.

Chapter 3 presents 38 dirty data types based on a set of business rules. A rule-based taxonomy of dirty data is given with these 38 dirty data types. The proposed taxonomy of dirty data is critically analysed and compared with existing research from the literature.

Chapter 4 presents a novel data cleaning framework. The components of the framework are detailed in this chapter and it is shown that the proposed framework retains the most appealing characteristics of the existing data cleaning approaches and enjoys being able to improve the efficiency/effectiveness of data cleaning in database applications.

Chapter 5 analyzed and evaluated a set of popular name matching algorithms on a set of carefully designed personal name datasets. The experimental results confirm the statement that there is no clear best technique. Suggestions regarding the selection of an appropriate name matching algorithm are presented, which can be used as guidance for researchers and practitioners to select an appropriate name matching algorithm for a given dataset.

Chapter 6 concludes the research and discusses the future work.

CHAPTER 2 LITERATURE REVIEW AND RELATED WORK

This chapter conducts a broad survey of many techniques that have been found useful during the data cleaning process as well as the improvement of data quality in database applications. For example, existing taxonomies of dirty data types from the literature will be reviewed to present the multiple dirty data types observed in different data sources. Data cleaning methods and approaches will also be reviewed in this chapter. They provide the foundation of the development of the proposed data cleaning framework. Data quality, data quality dimensions are reviewed in the final part of this chapter.

2.1 Dirty data

In Chapter 1, it is pointed out that many enterprises do not pay adequate attention to the existence of dirty data and have not applied useful methodologies to ensure high quality data for their applications. One of the reasons is a lack of appreciation of the types and extent of dirty data [6]. Therefore, in order to improve the data quality, it is necessary to understand the wide variety of dirty data that may exist within the data source as well as how to deal with them. In this section, current classifications or taxonomies of dirty data are reviewed first.

2.1.1 Müller and Freytag's Data Anomalies

In this work, the authors state that the definition of what is dirty data and what is not is highly application specific and have firstly presented the following definitions:

- Data: are symbolic representations of information, i.e., facts or entities from the world, depicted by symbolic values. They are collected to form a representation of a certain part of the world called the miniworld (M).

- Anomaly: is a property of data values that renders them a wrong representation of the miniworld.

With the above two definitions, the authors define ‘Data containing anomalies is dirty data’. According to the constraints specified in Müller and Freytag’s pre-defined data model, data from a data collection that does not conform to the constraints of the data model is considered as data anomaly. They roughly classify data anomalies into three different sets, each of which contains different dirty data types. The three different sets are called syntactical anomalies, semantic anomalies and coverage anomalies respectively [27].

Syntactical anomalies consider dirty data from data’s representation angle. There are three dirty data types, namely lexical errors, domain format errors and irregularities. Lexical errors show the difference between the structure of the data items and the specified format. Domain format errors specify that the given value for an attribute does not conform to the anticipated domain format. Irregularities deal with the problem of non-uniform use of values, units and abbreviations. Semantic anomalies mainly concern two types of dirty data: data that violates the integrity constraints and duplicate data. Integrity constraints are used to specify the rules for representing knowledge about the domain and the values allowed for representing certain facts. Duplicate data here stands for two or more tuples that represent the same entity. Finally, coverage anomalies describe the dirty data due to missing values or missing tuples. Apart from these 7 data anomalies, the authors also mentioned another data anomaly called invalid tuple, where data from the data collection conform to all the constraints of the data model but are still invalid entities from the mini-world. For example, a student’s age is 25 years old, but the value of age is entered as 26. Clearly, it is practically impossible for any software even a person to detect such an error [6]. Table 2.1 shows the dirty data types classified in this work.

No.	Dirty data type
MF.1	Lexical error
MF.2	Domain format error
MF.3	Irregularities error
MF.4	Integrity constraint violations error
MF.5	Duplicate records
MF.6	Missing values (null value not allowed)
MF.7	Missing tuple
MF.8	Invalid tuple

Table 2.1 Data anomalies from Müller and Freytag.

2.1.2 Rahm and Do's classification of data quality problems

Rahm and Do replace the term 'dirty data' by 'data quality problem'. According to the authors, database systems enforce restrictions of a specific data model as well as application specific integrity constraints. They distinguish the observed data quality problems into two sets namely single-source problems and multi-source problems. Within each set, data quality problems again have been classified into schema-level problems and instance-level problems respectively.

Within single-source problems, data quality problems occur due to the lack of appropriate model-specific or application-specific integrity constraints are defined as schema-level data quality problems. Data quality problems related to errors and inconsistencies that can't be prevented at the schema-level are defined as instance-level data quality problems [28].

Within multi-source problems, as different data sources are designed and maintained

independently, when these data sources are to be integrated, data quality problems become more complicated. The main problems at the schema-level are naming conflicts and structural conflicts. For example, one data source uses ‘Cid’ as the attribute name to represent customer identification number while in another data source, it may use ‘Cno.’ to represent the customer identification number, i.e., different names for the same attribute. As an example of structure conflicts, in one data source, attribute ‘name’ requires the name values to be written following the pattern as <given name last name>. Therefore, a person whose first name is ‘John’ and last name is ‘Smith’ will be written as “John Smith” in this data source according to the pre-defined pattern. However in another data source, name values may be required to be written in different attributes, e.g. ‘First Name’, ‘Last Name’ respectively. At the instance-level, problems may occur due to data conflicts such as different value representations or different interpretations of the same value . Furthermore, the authors also mentioned the existence of overlapping data that causes the problem of duplicate records within the multi data sources as well as contradicting records among multiple data sources. The dirty data types they introduced are shown in table 2.2.

No.	Dirty data type
RD.1	Illegal values due to invalid domain range
RD.2	Violated attribute dependences at schema level
RD.3	Uniqueness violation
RD.4	Referential integrity violation
RD.5	Missing values (null allowed)
RD.6	Cryptic values, Abbreviations
RD.7	Misspellings
RD.8	Embedded values
RD.9	Misfielded values
RD.10	Violated attribute dependences at instance level
RD.11	Word transpositions
RD.12	Duplicated records in single data source
RD.13	Contradicting records in single data source
RD.14	Wrong references
RD.15	Naming conflicts
RD.16	Structural conflicts
RD.17	Data conflicts in multiple data sources
RD.18	Duplicate records in multiple data sources
RD.19	Contradicting records in multiple data sources

Table 2.2 Dirty data types from Rahm and Do.

2.1.3 Kim *et al*'s taxonomy of dirty data

In this work, according to the authors, dirty data is defined roughly as either missing data or wrong data or non-standard representations of the same data [6]. Kim *et al* present a hierarchically structured taxonomy of dirty data. According to the different ways of dirty data manifestation, all dirty data that could be captured from different data sources can only be classified into the following three categories [6]:

- Missing data;
- Not missing but wrong data;
- Not missing, not wrong but unusable data;

These three categories of dirty data formed the main body of the taxonomy work. For the rest of the taxonomy work, the authors applied a hierarchical decomposition method to the three categories of dirty data and produced the taxonomy with 33 different types of dirty data.

Missing data is data that is missing in a field when it should not be missing. Two dirty data types are considered in this category: "missing data null value allowed" and "missing data null value is not allowed".

Not missing but wrong data is the data that is different from the "true value" of the data when it is accessed. In this category, dirty data is initially classified into two sets according to whether or not automatic enforcement of integrity constraints is available. For dirty data that can be prevented by automatic enforcement of integrity constraints, they can be classified based on whether these integrity constraints are supported by current relational database systems or these integrity constraints require extensions to current systems. For dirty data that can't be prevented by automatic enforcement of integrity constraints, the authors consider dirty data that arise in single-source and multi-source respectively. Together, there are 17 dirty data types within this category.

Not missing, not wrong, but unusable data is the data that is in some sense not wrong, but can lead to wrong results in a query or analysis. Dirty data that falls in this category is considered whether it arises in single-source or multi-sources respectively. In single-source, these data become dirty due to either the value of the data being ambiguous or the value of the data not conforming to standards. In

multi-source, it is due to the same entity having different values across multiple databases. Within this category, 14 dirty data types have been introduced. Table 2.3 shows the list of all 33 dirty data types from Kim *et al*.

Category	No.	Dirty data type
Missing data	K.1	Missing data (null value allowed)
	K.2	Missing data (null value not allowed)
Not missing, but wrong data	K.3	Use of wrong data type including value range
	K.4	Dangling data
	K.5	Violation of uniqueness constraint data
	K.6	Mutually inconsistent data
	K.7~K.10	Dirty data due to failure of transaction management facility
	K.11	Wrong categorical data
	K.12	Outdated temporal data
	K.13	Inconsistent spatial data
	K.14	Erroneous entry
	K.15	Misspelling
	K.16	Extraneous data
	K.17	Entry into wrong fields
	K.18	Wrong derived-field data from stored data
	K.19	Inconsistency across multiple tables/files due to integration constraint problem
Not missing, not wrong but	K.20	Different data for the same entity across multiple databases
	K.21	Ambiguous data due to use of abbreviation
	K.22	Ambiguous data due to incomplete context

unusable data	K.23	Different representation for non-compound data due to use of abbreviation (algorithms transformation is not possible)
	K.24	Different representation for non-compound data due to use of Alias/ nickname (algorithms transformation is not possible)
	K.25	Different representation for non-compound data due to encoding format (algorithms transformation is possible)
	K.26	Different representation for non-compound data due to different representations (algorithms transformation is possible)
	K.27	Different representation for non-compound data due to measurement units (algorithms transformation is possible)
	K.28	Different representation for compound data due to abbreviation
	K.29	Different representation for compound data due to use of special characters
	K.30	Different representation for compound data due to different ordering
	K.31	Different representation for hierarchical data due to abbreviation
	K.32	Different representation for hierarchical data due to use of special characters
	K.33	Different representation for hierarchical data due to different ordering

Table 2.3 Dirty data types from Kim *et al.*

2.1.4 Oliveira *et al*'s taxonomy of data quality problems

Oliveira *et al* also use the term 'data quality problem' to replace the term 'dirty data'. All problems that affect data quality are defined as data quality problems and the work aims at identifying all the data quality problems and organizes them according to a taxonomy. The taxonomy by Oliveira *et al* was formed by collecting the different data quality problems from the previous work [6, 27, 28]. These problems are structured under 6 different levels ranging from the lowest level problems (in a single attribute value of a single tuple) to the highest level problems (multi-source problems) [30].The six different levels are:

- L.1: *Problems related with an attribute value of a single tuple. (In single table of a single data source)*
- L.2: *Problems related with values of a single attribute. (In single table of a single data source)*
- L.3: *Problems related with multiple attribute values. (In single table of a single data source)*
- L.4: *Problems related with attribute values of several tuples. (In single table of a single data source)*
- L.5: *Problems related with relationships among multiple tables. (In multiple tables of a single data source)*
- L.6: *Problems related with multiple data sources.*

Compared with the earlier work, Oliveira *et al* present a rather complete taxonomy of data quality problems with 35 dirty data types presented. Table 2.4 shows the list of 35 data quality problems identified by Oliveira *et al*.

Level	Number	Dirty data type
L.1	O.1	Missing value
	O.2	Syntax violation for an attribute value of a single tuple in single data source
	O.3	Outdated value
	O.4	Interval violation
	O.5	Set violation
	O.6	Misspelled error
	O.7	Inadequate value to the attribute context
	O.8	Value items beyond the attribute context
	O.9	Meaningless value
	O10	Value with imprecise or doubtful meaning
	O.11	Domain constraint violation for an attribute value of a single tuple in single data source
L.2	O.12	Uniqueness value violation
	O.13	Synonyms existence
	O.14	Domain constraint violation for the values of a single attribute in single data source
L.3	O.15	Semi-empty tuple
	O.16	Inconsistency among attribute values
	O.17	Domain constraint violation for attribute values of a single tuple in single data source
L.4	O.18	Redundancy about an entity in single data source
	O.19	Inconsistency about an entity in single data source
	O.20	Domain constraint violation for attribute values of several tuple in single data source

L.5	O.21	Referential integrity violation
	O.22	Outdated reference
	O.23	Syntax inconsistency in single data source
	O.24	Inconsistency among related attribute values
	O.25	Circularity among tuples in a self-relationship
	O.26	Domain constraint violation for relationships among multiple relations in single data source
L.6	O.27	Syntax inconsistency in multi data sources
	O.28	Different measure units in multi data sources
	O.29	Representation inconsistency in multi data sources
	O.30	Different aggregation levels in multi data sources
	O.31	Synonyms existence in multi data sources
	O.32	Homonyms existence
	O.33	Redundancy about an entity in multi data sources
	O.34	Inconsistency about an entity in multi data sources
	O.35	Domain constraint violation in multi data sources

Table 2.4 Oliveira *et al*'s dirty data set.

A brief comparison among these four works mentioned above is given below:

Müller and Freytag [27] identify a set of errors (anomalies) that will affect data quality. The set includes lexical error, domain format error, irregularities, constraint

violation, missing value, missing tuple, duplicates and invalid tuple. Müller and Freytag's classification of anomalies does not present as many dirty data types as the other three works. This is because Müller and Freytag do not consider problems from multi-data sources. Their work limited the data quality problems to single data source. Rahm and Do [28] classify data quality problems into two groups: single-source and multi-source problems. However, at single-source, they do not divide the problems into those that occur in a single relation and those that occur in multi relations as Oliveira *et al* have done [30]. Kim *et al*'s work [6] presents a comprehensive taxonomy of dirty data, which is hierarchically structured. According to the different ways in which dirty data manifest, all dirty data that can be captured from different data sources are classified into the following three categories which form the main body of the taxonomy work. For the rest of the taxonomy work, the authors apply a hierarchical decomposition method to the three categories of dirty data and produce a taxonomy with 33 distinct dirty data types. Oliveira *et al* produce a very complete taxonomy [30]. They adopted a bottom-up approach, from the lowest level where data quality problems may exist (the ones that occur in a single attribute value of a single tuple) to the highest level (those that involve multi-source problems). At the single source level, problems are further divided into two sub-groups: those that occur in a single relation and those that result from existing relationships among relations. At the multi-source level, the data quality problems are decomposed into 9 problems. The work also proposed some dirty data types that Kim *et al* have not mentioned, e.g. DT.7, DT.13, DT.16, and DT.18. Although Oliveira *et al* provide the most comprehensive taxonomy compared with others, it still lacks of some dirty data types from them. For example, some dirty data types mentioned by Kim *et al* (DT.1, DT.19, DT.25, DT.34) are not included by Oliveira *et al*.

In this thesis, a rule-based taxonomy of dirty data is proposed. The dirty data is defined as 'the data flaws that break the data quality rules', which can be used to

measure the occurrence of data flaws. In Chapter 3, data quality rules will be discussed in detail and the proposed rule-based taxonomy of dirty data will be presented. A comparison between the proposed taxonomy with these research works described above will be given. It will be shown later that the proposed taxonomy of dirty data not only provides a solution to deal with the DDS problem but also includes more dirty data types than any of the existing taxonomies.

2.2 Methods used for Data cleaning

In this section, general existing methods or techniques that could be used for data cleaning tasks are reviewed. They are developed to deal with some popular data cleaning activities exclusively and have been implemented into some of the existing commercial data cleaning tools.

(1) Parsing

Parsing in data cleaning is performed for the purpose of detecting syntax errors. Parsing decides for a given string whether it is an element of the language defined by the correct grammar. For example, the framework of Potter's Wheel provides two mechanisms for the parsing task namely 'Type-based Discrepancy Detector (TDD)' and 'User-specified Discrepancy Detector (UDD)' [33]. A TDD is an algorithm which detects discrepancies in values of a particular type. A UDD is a discrepancy detection algorithm that the user asks the system to apply on a specific set of fields [33]. As an example of using the two types of parser, suppose the schema of the table containing student records of a university in the U.K. is represented with: Student={StudentName, DepartmentName, StudentID, DateofBirth}, and suppose the user has registered a Number TDD that maintains as internal state the mean and standard deviation of values seen so far, and flags any value that is more than 10 standard deviations from the mean as dirty data. The user has also registered a String

TDD that matches strings of alphabets and the user has specified a UDD on the DepartmentName column to check validity of department name. Suppose a student record with values {John, Computing, 001, 01/01/1988} is chosen to extract structures (parsers). The result of the inferred parsers will be:

Name:*String*,
DepartmentName:*String*,
StudentID:*Number*,
DateofBirth:*Number/Number/Number*.

In this case, the following records will be detected as the ones containing dirty data:

Record 1: {Sally, HelloWorld, 002, 05/06/1988}
Record 2: {Jack, Math, 004, March, 4, 1986}
Record 3: {Tom, Computing, 005, 09/10/19827}

In Record 1, according to the DepartmentName UDD, the value 'HelloWorld' will be detected as an anomaly since the value violates the valid department names defined in the DepartmentName UDD. In Record 2, the value of DateofBirth 'March, 4, 1986' will be detected as an anomaly since its structure is extracted as '*String, Number, Number*' and violates to the registered TDD for the DateofBirth field. Finally, in Record 3, the Number TDD will be invoked on the sub-components of DateofBirth field and will detect the value of year '19827' as an anomaly because it is too many deviations away from the mean value of the year sub-component.

(2) Data transformation

Data transformation is one of the major subtasks in data preparation. It transforms the data on a structural level as well as an instance level, meeting the requirements

of the analysis tools. Although there are many commercial tools available for the transformation problems such as Microsoft Data Transformation Service or Oracle Data Warehouse Builder, they perform transformation in a batch-like manner not supporting an explorative and interactive approach. The solution of using a multi-database query language (FRAQL) helps with improve this shortcoming, as it is possible to check various strategies for integration and cleaning with reduced effort [34].

According to Sattler and Schallehn, FRAQL provides good solutions for data transformation problems on both the schema and instance level. On the schema-level, two operations namely “TRANSPOSE TO ROWS” and “TRANSPOSE TO COLUMNS” are provided by FRAQL, which help with converting rows to columns and vice-versa. On the instance-level, FRAQL can help with simple value conversions as well as attribute value normalizations. The simple value conversions can be realized with the built-in functions such as ‘string manipulation functions’ or ‘general purpose conversion functions’. Attribute value normalizations can be realized with the user-defined functions (UDF), which help to normalize the attribute values to lie in a fixed interval given by the minimum and maximum values [34].

(3) Integrity constraint enforcement techniques

Database integrity refers to the validity and consistency of stored data. Integrity is usually expressed in terms of constraints, which are consistency rules that the database is not permitted to violate. Techniques for integrity constraint enforcement can help with eliminating integrity constraint violation problems. In general, integrity constraint enforcement ensures the satisfaction of integrity constraints after transactions modifying a data collection by inserting, deleting, or updating tuples have been performed. There are two approaches, namely integrity constraint

checking and integrity constraint maintenance respectively. The former will help to prevent the occurrence of integrity constraints during a transaction. The latter will help to correct integrity constraint violation after the transactions in order to guarantee that the resulting data collection be free of integrity constraint violations. According to Maletic and Marcus, integrity analysis can be used to locate data errors [21]. Given a dataset that adheres to the relational model, the data integrity analysis can be used as a simple data cleaning operation. Relational data integrity, including entity, referential, and column integrity can be accomplished using relational database queries such as SQL [21]. However, limitations exist in applying these integrity constraint enforcement techniques, e.g., the control of the data cleaning process must remain with the user all the time and it can only uncover a number of possible errors in a data set but not some more complex problem such as outlier detection.

(4) Duplicate detection techniques

Duplicate detection or record matching is an important process in data cleaning. It involves identifying whether two or more tuples are duplicate representations of the same entity. Duplicate records do not share a common key and contain erroneous data that make record matching a difficult task. There are two main approaches for duplicate record detection, categorized into two approaches: approaches that rely on training data, e.g., probabilistic models [10] or supervised/semi-supervised learning techniques [35-39] and approaches such as rule-based [13, 40, 41] and distance-based techniques [42-44] that rely on domain knowledge or distance metrics to match records.

With respect to the former approach, the limitation exists that training data may not always be available all the time. Although the unsupervised ‘Expectation Maximization’ (EM) algorithm is available to supply the maximum likelihood

estimate, there are some conditions required in order to use the EM algorithm. For example, the rate of typographical error should be low and there should be more than 5% duplicates within the dataset. On the other hand, the rule-based approach does not require the training data. However, an expert is needed for devising the matching rules set in order to obtain high accuracy of matching result. Therefore, a limitation of this approach exists in that an expert may not always be available all the time and the rules set may be domain specific. In addition, distance-based algorithms are needed to be applied for the rule-based approach. For example, an approximate string matching algorithm such as the Jaro algorithm or Levenshtein algorithm may be applied on the name strings values between two records to determine the degree of similarity between the two records with the help of a pre-defined threshold value. A poor selection of the threshold value will generate a poor matching result. Therefore, the choice of a proper distance-based algorithm and a selection of a suitable threshold value play an important role in the rule-based approach. Some popular character-level string matching algorithms will be analyzed in chapter 5 as well as the threshold value selection problem.

(5) Statistical methods

Some statistical methods can be used for auditing data as well as correcting erroneous data. For example, Marcus shows that statistical methods can help with identifying outlier problems using the values of mean, standard deviation, range based on Chebyshev's theorem, considering the confidence intervals for each field [21]. In this work, outlier values for particular fields are identified based on automatically computed statistics. For each field, the average and the standard deviation are utilized and based on Chebyshev's theorem. Those records that have values in a given field outside a number of standard deviations from the mean are identified. The number of standard deviations to be considered is customizable. Confidence intervals are taken into consideration for each field.

2.3 Existing approaches for Data cleaning

In chapter 1, the data cleaning process is described as a complex process with massive human resource involvement. Section 2.2 discussed general techniques and methods that can be used to automate activities in data cleaning applications as far as possible. In this section, five selected data cleaning approaches from the literature are reviewed.

(1) Potter's Wheel

Potter's Wheel is an interactive framework for data cleaning and transformation. According to the authors, data often has inconsistencies in schema, formats, and adherence to constraints. This may be due to many factors such as data entry errors or data integration from multiple data sources. Therefore, data that do not conform to the required formats either on the instances level or schema level must be detected and transformed into a uniform format before using it.

Although many data cleaning tools exist when Potter's Wheel was developed, according to Raman and Hellerstein, those tools had serious drawbacks in usability. The main drawbacks include (1) these tools use a combination of analysis tools and transformation tools together to deal with the discrepancy detection and the data transformation respectively, with little interactivity, (2) the detection of discrepancy and data transformation are typically performed within a batch process, operating on a table or the whole database, without any feedback. Users have to face long frustrating delays and they will have no idea if a transform is effective and (3) some 'nested discrepancies' are hard to detect in only one pass. Therefore, more iterations are required between discrepancy detection and data transformations since users have to wait for a transformation to finish before they can check if it has fixed all anomalies.

Potter's Wheel was developed to address these drawbacks. According to the authors, any data cleaning solution must support transformation and discrepancy detection in an integrated fashion. Regarding the transformation, it must be general and powerful enough to do most tasks without explicit programming. However, some commercial extract, transform and load (ETL) tools typically only support some restricted transforms between a small set of formats via a graphical user interface (GUI). Regarding discrepancy detection, it must support the variety of discrepancy detection algorithms applicable in different domains. However, the techniques applied in some data auditing tools for the purpose of discrepancy detection are domain specific, which is unsuitable for detecting the data composite structures of values from different domains. Users have to either write a custom program for each such structure or design transforms to parse data values into atomic components for anomaly detection. Finally, transformation and discrepancy detection should be realized through simple specification interfaces and within minimal delays. Potter's Wheel is just such an interactive data cleaning system that integrates transformation and discrepancy detection in a single interface. The following figure shows Potter's Wheel's architecture [33]:

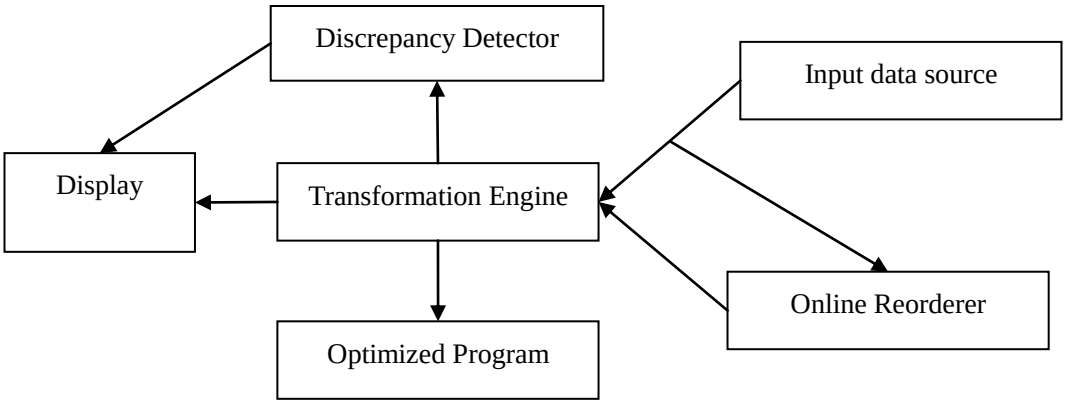


Fig.2.1 Potter's Wheel Architecture

The main components of Potter's Wheel (Fig.2.1) are 'Online Reorderer', 'Transformation Engine', and 'Discrepancy Detector'. The 'Online Reorderer' is a

feature exclusively designed for Potter's Wheel. It fetches tuples from the 'Input data source' continually and divides them into buckets, spooling them to disk if needed. The 'Online Reorderer' picks a sample of tuples from the bucket corresponding to the scrollbar position and displays them on screen. This allows users to interactively resort on any column and scroll in a representative sample of the data, even on large datasets. The 'Transformation Engine' deals with the different transforms such as schema level data transforms which can help with splitting one data field into several fields or combining different fields into one single data field, or instance level data conflicts associated with the discrepancy detection task. Traditionally, some common transformations can be realized without explicit programming, they have been used in some commercial ETL tools. However, some transforms require parsing and splitting values into atomic components, which are quite complex and require users to write custom programs. In Potter's Wheel, a 'structure extraction technique' is developed exclusively to automatically infer patterns in terms of different domains. This enables users to specify the desired results on the example values and automatically infers a suitable transform. The 'Discrepancy Detector' runs in the background when a transform is specified and data is explored. Appropriate algorithms specified for different domains will be applied to detect data anomalies. In Potter's Wheel, the transforms are specified graphically and their effects are shown immediately on records visible on screen. If their effects are undesirable, undone can be performed easily. At the same time, discrepancy detection is done automatically in the background based on the latest transformed view of the data. The detected anomaly will be flagged. In this way, users can gradually develop and refine transforms as the discrepancies are found. After constructing a satisfactory sequence of transforms, the user can ask the system to generate an optimized program to run on the dataset as a batch, unsupervised process.

Although the ability of user interactivity is improved with the help of the 'Online

Reorderer' exclusively developed in Potter's Wheel, the degree of automation of Potter's Wheel is low as the detection of the data that requires data transforms is totally depending on the manual perception. This is a limitation regarding the efficiency of performing data cleaning tasks. Besides, since Potter's Wheel is mainly focused on solving the data transformation problems either on the instance-level or scheme-level, problems such as duplicate record detection is not supported well in Potter's Wheel and users need to seek other tools to deal with duplicate record detection problem.

(2) AJAX

The main goal of AJAX is to facilitate the specification and execution of data cleaning programs either for a single data source to help with dealing with duplicate record detection problem, or for integrating multiple data sources into a single new data source [41].

Although some existing ETL tools provide platforms to implement some data transformations, the drawback according to the authors, is that they lack a clear separation between the logical specification of data transformations and their physical implementations. The solution for some tools only consists of a specific optimized algorithm which is already parameterized with some user provided criteria. This can't fit all situations. Besides, the user interaction facilities in these tools are poor. Sometimes, an expert consultation is required during a data cleaning process. For example, when two different publication date values for the same published work are detected, the one to keep requires a judgement from the user. However, in existing tools, there is no specific support for user consultation except to write the data to a specific file to be analyzed by the user later.

AJAX was developed as a data cleaning framework which attempts to separate the data cleaning program into two levels namely the logical level and the physical level. The logical level supports the design of the data flow graph that specifies the data transformations needed to clean the data. These data transformations are specified with four main logical operators namely, mapping, matching, clustering, and merging. The mapping operator standardizes data formats when necessary. For example, it can convert the name string values into lower case. It can also help with producing records with a more suitable format by applying operations such as column splitting and merging. For example, values in an 'address' field can be split into separated address components such as 'city', 'street', 'number'. The matching operator finds pairs of records that most probably refer to the same real object. The clustering operator groups together the matching pairs with high similarity values with the help of a given grouping criteria, e.g., transitive closure. The merging operator is applied to each individual cluster returned by the clustering operator to eliminate duplicate records or produce new records. The design of these operators is based on the semantics of SQL primitives which are extended to support a larger range of data cleaning transformations. For example, Fig.2.2 and Fig.2.3 show the use of the mapping and matching operators respectively.

```
CREATE MAPPING MP
SELECT s.key, lowerName, city, street, number
FROM SUBSCRIBERS s
LET IF (s.name==null) throw NullException(s.key)
                lowerName=lowerCase(s.name)
                [city, street, number]=extractAddressComponents(s.address)
```

Fig.2.2 Mapping operator from AJAX

In Fig.2.2, the mapping operator converts names into lower case and the address field is split into separate components. An exception is raised if the name field is null and a human expert is called later. In SQL, an exception will immediately stop

the execution of a query. The semantics provided in AJAX enables computing the entire set of tuples regardless of the exceptions.

```
CREATE MATCHING M1
FROM GSM-CLIENT g1, GSM-CLIENT g2
LET similarity=nameSIMF(g1.name, g2.name)
WHERE g1.gsmID<g2.gsmID
AND similarity>0.5
    {SELECT g1.gsmID AS gsmID1, g2.gsmID AS gsmID2, similarity AS
                                        similarity
KEY gsmID1, gsmID2}
```

Fig.2.3 Matching operator from AJAX

In Fig.2.3, the matching operator is used for finding duplicate records within a data source called GSM-CLIENT. In this example, an approximate string matching algorithm, i.e., nameSIMF() is applied to compute the similarity between the two name values and a threshold value 0.5 is used to classify the matching results. There are many approximate string matching algorithms available for different types of strings concerning the different domains involved. They will be further discussed in chapter 5.

In the logic level, the main constituent of a data cleaning program is the specification of a data flow graph where nodes are the logical operators. Each operator can make use of externally defined functions or algorithms that implement domain specific treatments such as extracting substrings from a string, computing the distance between two string values, etc. A feature exclusively designed for these operators is the automatic generation of a variety of exceptions for each operator. For each exception thrown, the corresponding information of the data item is then stored with a textual description of the exception. A data lineage mechanism enables users to inspect exceptions, analyze their provenance in the data flow graph and

interactively correct the data items that contributed to its generation. The corrected data can then be re-integrated into the data flow graph. In this way, user interaction is enforced.

The physical level supports the implementation of the data transformations and their optimizations. The focus here is the design of performance heuristics that can improve the execution speed of data transformations without sacrificing accuracy. Although the physical level can help with selecting an efficient algorithm to implement a logical operation among a set of alternatives, it is the users who control the proper usage of optimization algorithms in the logical level. For example, suppose a matching task to deal with duplicate record detection problem is required during the data cleaning process. Users have to specify the information such as operators involved the properties of the matching algorithms, the required parameters for optimization in the logical level. In the physical level, the system then will consume the information obtained from the logical level and then specific optimized algorithms can be selected to implement the transformations. This, however is a limitation regarding the effectiveness during the data cleaning process. As will be discussed later, a poor setting of the required parameters for the selected technique will generate poor matching results. For example, when an approximate string matching technique is selected for the matching of records, how to set the threshold value is still unclear. Usually, a universal value will be chosen for all situations in AJAX. As will be seen later in chapter 5, many factors are needed to be considered when setting the threshold value for the selected matching algorithm in order to achieve a better matching result. However, in AJAX, the proper usage of the optimization algorithm in the logical level entirely depends on its users without considering any different factors such as problem domains.

(3) ARKTOS

According to the developers of ARKTOS, in the context of a data warehouse, both schema and instance levels should be considered during the integration of data [45]. Although there are some tools such as some commercial ETL tools as well as data cleaning tools existing to help with data integration, they are responsible for parts of tasks such as the extraction of data from several sources, or for cleaning a specific dirty data type exclusively. This makes the use of these tools complex and pricy. Therefore, ARKTOS was developed as a data cleaning tool with the following goals: (1) the data warehouse transformations and the data cleaning tasks can be defined with graphical and declarative facilities, (2) the quality of data can be measured with specific quality factors, and (3) the complex sequence of transformation and cleaning tasks could be optimized.

In ARKTOS, for each dirty data type, the detection of dirty data is performed by an ‘activity’. An activity is an atomic unit of work and a discrete step in the chain of data processing. The work performed by each activity is specified by an SQL statement, which gives the logical, declarative description of the work. Each activity is accompanied by an error type and a policy. An error type of an activity identifies the problem the process is concerned with such as ‘Primary key violation’, ‘NULL value existence’, etc. A policy signifies the way the data should be treated such as ‘deleting the tuples’, ‘reporting the tuple to a file or table’. When multiple activities are involved in ARKTOS, the users can tailor the set of activities to be executed all together with the help of a ‘scenario’ (a set of processes to be executed all together) defined in ARKTOS.

In ARKTOS, the error types the system can deal with include (i) primary key violation, (ii) reference violation, (iii) null value existence, (iv) uniqueness violation, (v) domain mismatch, and (vi) field format transformation. Two methods are

proposed in ARKTOS to specify each activity either graphically or declaratively. These two methods overcome the issues of user-friendliness and complexity of the exiting ETL tools mentioned in the beginning. Regarding the graphical method, a palette with all the possible activities provided by ARKTOS is available for user to compose a scenario from these activities and link them in a serial list to execute. Regarding the declarative method, two declarative definition languages are proposed by ARKTOS namely 'XML-based Activity Definition Language' (XADL) and 'Simple Activity Definition Language' (SADL) respectively.

XADL is an XML language for data warehouse processes, on the basis of a well-defined DTD, writing of SADL is verbose and complex but is more comprehensible. SADL is a declarative definition language motivated from the SQL paradigm, it is more compact and resembles SQL and is suitable mostly for the trained users. Fig.2.4 and Fig.2.5 show an example proposed by the authors how the two languages used for a specification of a scenario in ARKTOS [45]. The scenario in this example tries to solve the following activities ordered as follows: (1) Push data from table LINEITEM of source database S to table LINEITEM of the DW database. (2) Perform a referential integrity violation checking for the foreign key of table LINEITEM in database DW, which is referencing table ORDER. Delete violating rows. (3) Perform a primary key violation check to the table LINEITEM and report violating rows to a file.

Similar to Potter's Wheel, the dirty data types mainly addressed in ARKTOS are schema-level and instance-level data transformations as well as some integrity constraints enforcement. ARKTOS can't deal with duplicate record detection problem as AJAX.

Although ARKTOS allows specifying a set of data cleaning tasks to be performed as a 'scenario', the 'scenario' is composed by its users without providing any detailed

information as how such a scenario should be composed when considering the multiple factors involved during the data cleaning process such as the different problem domains, algorithms involved etc. The developers of ARKTOS do not give any further investigations on the ‘ordering’ problem when multiple cleaning tasks are associated during the data cleaning process.

```
1. <?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
...
67. <transformtype>
68.       <input_table table_name="lineitem" database_url="jdbc:informix-sqli:
                  //kythira.dbnet.ece.ntua.gr:1500/dbs3:informixserver=ol_milos_tcp">
69.       <column> l_orderkey </column>
70.       <column> l_partkey </column>
...
85.       </input_table>
86.       <errortype>
87.           <reference_violation>
88.               <target_column_name> l_orderkey </target_column_name>
89.               <referenced_table_name> Informix.tpcd.tpcd.tpcd.order </referenced_table_name>
90.               <referenced_column_name> o_orderkey </referenced_column_name>
91.           </reference_violation>
92.     </errortype>
93.     <policy> <delete/> </policy>
94.     <quality_factor qf_name=No_of_reference_violations qf_report_file="H:\path\scenario3.txt">
95.           <sql_query> select l_orderkey from lineitem t1 where not exists
                        (select o_orderkey from order t2 where t1.l_orderkey = t2.o_orderkey)
            </sql_query>
96.     </quality_factor>
97. </transformtype>
...
140.</scenario>
```

Fig.2.4 XADL definition of a scenario, as exported by ARKTOS

```
1. CREATE SCENARIO Scenario3 WITH
2. CONNECTIONS S3,DW
3. ACTIVITIES Push_lnitem, Fk_lnitem, Pk_lnitem
4. ...
5. CREATE CONNECTION DW WITH
6. DATABASE "jdbc:informix-sqli://kythira.dbnet.ece.ntua.gr:1500/
                dbdw:informixserver=ol_milos_tcp" ALIAS DBDW
7. DRIVER "com.informix.jdbc.IfxDriver"
8. ...
9. CREATE ACTIVITY Fk_lnitem WITH
10. TYPE REFERENCE VIOLATION
11. POLICY DELETE
12. SEMANTICS "select l_orderkey from lineitem@DBDW t1 where not exists
                (select o_orderkey from order@DBDW t2 where t1.l_orderkey=t2.o_orderkey)"
13. ...
14. CREATE QUALITY FACTOR "# of reference violations" WITH
15. ACTIVITY fk_lnitem
16. REPORT TO "H:\path\scenario3.txt"
17. SEMANTICS "select l_orderkey from lineitem@DBDW t1 where not exists
```

Fig.2.5 SADL definition of a scenario, as exported by ARKTOS

(4) IntelliClean

IntelliClean is a knowledge-based framework for intelligent data cleaning which mainly deals with duplicate records elimination [9]. According to the authors, although domain knowledge plays an important part in data cleaning, little on knowledge management issues has been undertaken such as the representation of the domain knowledge used for data cleaning. Besides, traditional data cleaning methods used for duplicate detection depend on the basis of computing the degree of similarity between the nearby records in a sorted database. In this case, a recall-precision dilemma exists that high precision is achieved at the cost of lower recall. In order to address these problems, IntelliClean was developed as a framework which provides a systematic approach for representation standardization, duplicate elimination, anomaly detection and removal in dirty databases. Three

stages are included in this framework: (1) pre-processing stage, (2) processing stage, and (3) validation and verification stage.

In pre-processing stage, data anomalies such as domain constraint violations, misspelling and, inconsistent use of abbreviations, are firstly detected and cleaned. For example, date values such as '2/3/2011', 'March, 2, 2011' can be standardized into one format. The values like '1', 'A', 'M' in the gender field will all be replaced by the value of 'Male'. This can be realized with the help of some reference functions and look-up tables. These conditioned data records then will be input to the processing stage.

In the processing stage, the conditioned records are fed into an expert system engine together with a set of rules which are designed to help with detecting the duplicate records. Particularly, a new method to compute the transitive closure is proposed in IntelliClean to increase the recall. In IntelliClean, the 'knowledge-base' is formed by different rules generally written as the following form:

IF <condition> THEN <action>

These rules are derived naturally from the business domain. When the condition part of the rule is satisfied, the action part of the rule will be activated. The business analyst with subject matter knowledge is expected to fully understand the governing business logic and can develop the appropriate conditions and actions.

In IntelliClean, rules are fed into an expert system engine, making use of an efficient method for comparing a large collection of rules to a large collection of objects. According to the authors, simple rules may be generated automatically when supplied with necessary parameters. However, hand-coding might be required when more complex rules are needed.

All rules from IntelliClean can be categorized into four types namely ‘duplicate identification rules’, ‘merge/purge rules’, ‘update rules’ and ‘alert rules’ respectively. More specifically, duplicate identification rule specifies the conditions for two records to be classified as duplicates. For example, Fig.2.6 shows an example of the duplicate identification rule in IntelliClean. This example shows the duplicate records are searched in a restaurant relation, with attributes ID, Address, and Telephone.

In order to activate the rule specialized in Fig.2.6, the corresponding conditions must be satisfied: the telephone numbers must be matched, and one of the identifiers must be a substring of the other. In addition, the address values of the two records must be very similar with a similarity higher than 0.7 according to the selected function (FieldSimilarity). Records classified as duplicates with this rule will have a certainty factor of 70%. A certainty factor (CF) represents expert confidence in the rule effectiveness in duplicate record detection, where $0<CF<1$. A higher CF value can be assigned to a rule if it is sure that the rule will identify true duplicates.

The merge/purge rules specify how duplicate records are to be handled. For example, a simple rule might be like ‘only the tuple with the least number of empty fields is to be kept in a group for further analysis and delete the rest of the tuples.’ Update rules specify the way data is to be updated in a particular situation. For example, it can specify when value in a field of a tuple is missing, what value will be filled. Finally, an alert rule helps with raising an alert when certain events occur such as integrity constraint violations.

```
Define rule Restaurant_Rule
Input tuples: R1, R2
IF (R1.telephone=R2.telephone)
AND (ANY_SUBSTRING (R1.ID, R2.ID)=TRUE)
AND (FIELDSIMILARITY(R1.ADDRESS,R2.ADDRESS)>0.7)
THEN DUPLICATES(R1, R2) CERTAINTY=0.7
```

Fig.2.6 An example of the duplicate identification rule in IntelliClean

In the validation and verification stage, human intervention is required to manipulate the duplicate records which are not dealt with due to the lack of merge/purge rules. It also helps with the validation of the rule base. Any rule that generates a wrong result will be taken out or have its parameters changed. According to the authors, well-developed rules are effective in identifying true duplicate records but are strict enough to keep out similar records which are not duplicates. In this way, higher recall is achieved with more rules. As concluded by the authors, the recall increases with the number of rules, and more complex rules identified more true duplicate records. This helps with resolving the recall-precision dilemma problem mentioned in the beginning. For example, in IntelliClean, the sorted neighbourhood method (SNM) is used for the detection of duplicate records. After the running of this algorithm, transitive closure is computed to group the duplicate records. This procedure can raise the false positive error as incorrect pairs are merged and the precision of the result will be lowered.

IntelliClean tries to reduce the number of wrongly merged duplicate groups by applying a certainty factor (CF) to each duplicate identification rule. Fig.2.6 shows an example that a CF=0.7 is added for the pairs of tuples R1 and R2. During the computation of the transitive closure, the value of CF is compared to the user-defined threshold value. Any merges that result in a CF value less than the

threshold value will not be executed. In this way, the false positive error is lowered. One limitation for IntelliClean is that, according to the developers, only the method of SNM is supported in this tool to detect duplicate records. SNM is a good method to deal with duplicate record detection in large datasets. However, when small datasets are involved, clearly a pair wise comparison is the best way to improve the effectiveness.

(5) Febrl

Matching records that refer to the same entity across databases is becoming an increasingly important part of the process of data cleaning. Data from multiple sources needs to be matched in order to enrich data or improve its quality. Although significant advances in record linkage techniques have been made in recent years, according to the authors, the vast majority of them are a ‘black box’ commercial software because the details of the technology implemented within the linkage engine of these tools are normally not accessible. This makes it difficult for both researchers and practitioners to experiment with new record linkage techniques, and to compare existing techniques with new ones. Additionally, many of these tools are developed exclusively for a certain domain such as dealing with business data or dealing with customer mailing lists. For many applications, the record linkage may often involve dealing with data from heterogeneous sources from different domains. In this case, the record linkage task is often limited by the functionality provided by these tools. In order to address these drawbacks, a freely extensible biomedical record linkage system (Febrl) was developed, which contains many recently developed techniques for data cleaning, de-duplication and record linkage, and encapsulates them into a GUI [46].

For users who have limited programming experience, this tool helps with facilitating the use of record linkage techniques without the need of any programming skills.

Particularly, it is suitable for the rapid development, implementation, and testing of novel data cleaning, record linkage and de-duplication techniques due to the availability of its source code and it allows researchers to compare various existing record linkage techniques with their own ones, enabling the record linkage research community to conduct their work more efficiently.

According to the authors, Febrl is an open source data cleaning toolkit and the only freely available data cleaning, de-duplication and record linkage system with a graphical user interface (GUI) [47]. The Febrl system has been developed with a focus on the cleaning and linking of health related data. However, the techniques developed and implemented in Febrl are general enough to be applicable to data from a variety of other domains. Since it was first published in the early September 2002, the Febrl system has been hosted on the Sourceforge.Net open source software repository and is available from:

https://sourceforge.net/projects/febrl/

The latest version of the Febrl system is Febrl-0.4.2 released on December, 14, 2011. Febrl system mainly supports three types of projects namely ‘Standardization’, ‘Deduplication’ and ‘Linkage’ respectively. Fig.2.7 shows a screen-shot of the main Febrl GUI after start-up.

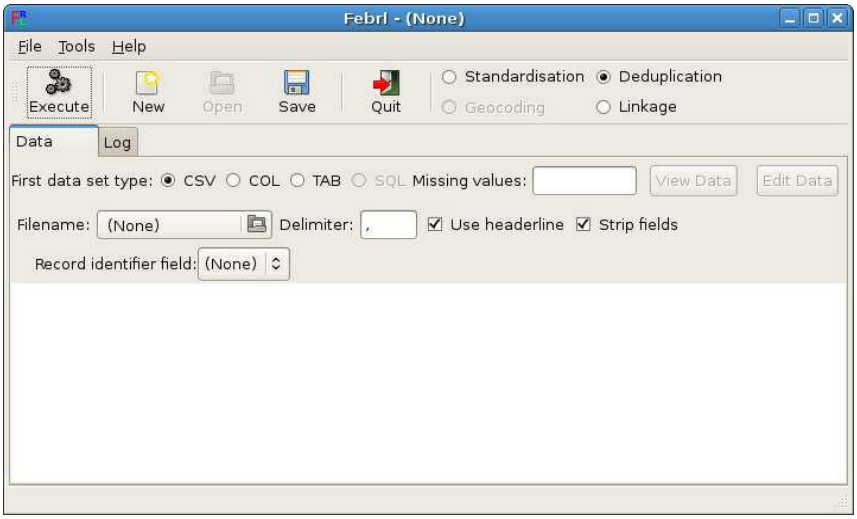


Fig.2.7 Initial Febrl user interface

In the middle top part of the Febrl GUI, the user can select the type of project he or she wants to conduct. The running of these three projects helps with finishing the record linkage process, which are detailed briefly as follows:

(a) Data cleaning and standardisation

In order to have a successful record linkage result, pre-processing of the input data is required. Regarding the input data, currently Febrl supports three types of text file formats: comma separated values (CSV), tabulator separated values (TAB), and column oriented values with fixed-width fields (COL). Access to a database is not supported in Febrl at the moment. The linkage process is usually based on the available record fields (attributes) such as personal names, address values, date of birth, etc. Values in such fields however often contain noisy, incomplete and incorrectly formatted information. Cleaning and standardization of these data therefore are an important first step for a successful record linkage.

The objective of this step is to convert the raw input data into the well-defined, consistent formats and resolve the inconsistencies in the raw input data. A running of

the ‘Standardisation’ project helps with this step. In this project, users can standardize the data from a selected file and then save the standardized data into a new file for the purpose of running a ‘Linkage’ or a ‘Deduplication’ project. In ‘Standardisation’ project, users can define one or more component standardisers. Currently, Febrl contains standardisers for names, addresses, dates, and telephone numbers. For each standardiser, a user needs to select one or several input fields from the input dataset and the user is required to supply the expected formats for its output fields. Additionally, all parameters for each standardiser are required to be set by the user.

(b) Matching and classification

During the matching process, potentially, each record in one dataset needs to be compared with all records in another dataset if a ‘Linkage’ project is selected or with the other records in the same dataset if a ‘Deduplication’ project is selected. This comparison process is therefore of quadratic complexity. In order to improve the scalability of the matching process, the potentially very large number of record pairs that are to be compared has to be reduced. This can be realized by some indexing techniques which split the databases into blocks. Only records that are in the same block are compared with each other with the help of the selected comparison functions such as Jaro, Levenshtein, Q-Gram, etc to the contents of the record fields (attributes). Several indexing techniques are provided in both ‘Deduplication’ and ‘Linkage’ such as the ‘FullIndex’ technique, the ‘BlockingIndex’ technique, and ‘SortingIndex’ technique. Once an indexing technique is selected, the actual index keys and their parameters have to be defined and provided by the user.

Regarding the comparison functions, Febrl provides 26 similarity functions for users to choose from. For each of these functions, users need to select two fields for comparison. Broadly, these functions can be categorized into two groups: functions

used for approximate string comparisons and functions used to compare fields containing numerical values such as age, date, and postcode. Finally, the compared records are classified into different groups such as match, non-match, and possible-matches groups. This is realized by applying different decision models against on the weight vectors obtained from the matching process.

However, unlike the function provided by IntelliClean, the merging of those linked records is not supported in Febrl and users have to merge the detected records manually.

Still, some limitations are observed in Febrl. Regarding the dirty data types addressed in Febrl, both data standardization and duplicate record detection are supported in Febrl. However, unlike other tools such as AJAX or IntelliClean, data standardization and duplicate record detection can not be specified within the same data cleaning process. Each data cleaning task should be specified and executed respectively. With respect to the data standardization, currently, Febrl only supports some limited instance-level data transformations. Unlike Potter's Wheel, further dirty data detection such as outlier detection against on the transformed data values is not supported in Febrl. Although Febrl supports a variety of techniques to deal with duplicate record detection, choosing a suitable technique as well as setting the corresponding parameters for the selected techniques entirely depends on its users. Febrl does not supply any recommendations or helps during the selection. For users who do not have any knowledge about these techniques, the use of Febrl is difficult. As will be seen later, even for users who are familiar with these techniques, a poor setting of the required parameters for the selected technique will generate poor matching results. Additionally, Febrl does not support a flexible merging towards the linked records after the detection of duplicate records, which make it harder to analyze the results. At the moment, Febrl only supports three types of text file formats as the input data: CSV, TAB, and COL. Loading input data from a database and write the linked output data back into a database is not supported in Febrl.

Finally, the installation of Febrl is quite complex which requires the manually installation of various Python modules. The execution of its techniques is slow. For large datasets, Febrl requires large amounts of memory which will result in a poor scalability.

(6) Summary

To sum up these five approaches, table 2.5 is provided. The names of the data cleaning approaches, the main activities addressed by these data cleaning approaches, and the special features associated with the five approaches are detailed in this table.

Name	**Activities**	**Special features**
Potter's Wheel	Schema-level data transformation Instance-level data transformation Domain constraint resolution	Tightly integrates transformations and dirty data detection Structure extraction technique
AJAX	Schema-level data transformation Instance-level data transformation Duplicate record detection	A separation of logical and physical plan for data cleaning
ARKTOS	Schema-level data transformation Instance-level data transformation Integrity constraints enforcement	A graphical method for user to specify a set of cleaning tasks
IntelliClean	Instance-level data transformation Domain constraint resolution Duplicate record detection	A recall-precision dilemma resolution
Ferbl	Instance-level data transformation Duplicate record detection	Open source software A graphical method for user to deal with data standardization and duplicate record detection.

Table 2.5 Summary of the five approaches

In detail: The main focus of Potter's Wheel is to stress user friendliness and interactivity in various data transforms and conflict resolution, resulting in tight integration of transformation and discrepancy detection.

The exclusive 'Online Reorderer' helps with realizing the ability of user interactivity. The 'Online Reorderer' continually fetches tuples from the data source and divides them into buckets. Each time, the 'Online Reorderer' only picks a sample of tuples from the bucket corresponding to the scrollbar position and displays them on the screen. Since the number of rows that can be displayed on screen at a time is small, users therefore can perceive any data transformations needed either on the schema-level or instance-level instantaneously. In this way, the user can perform the data transforms as they explore the data with the help of the 'Online Reorderer'.

While the user is specifying transforms and exploring the data, the discrepancy detector runs in the background and applies appropriate algorithms to detect errors in the transformed data fetched directly from the 'Online Reorderer'. Regarding the function provided for data transformation, Potter's Wheel allows users to specify the desired results on example values and automatically infers a suitable transform using the 'structure extraction technique' exclusively developed for Potter's Wheel. It allows users to define custom domains and have corresponding algorithms to enforce the domain constraints. Compared with other tools such as 'AJAX', 'IntelliClean' which only support some predefined domain specific transformations for the fields such as 'date of birth', 'telephone number', this is an advance. However, since the detection of the required data transformations in Potter's Wheel totally depends on the manual perception, the efficiency and the degree of automation is very low in this way compared with other tools.

The design of AJAX are twofold: (1) a declarative language for expressing data cleaning tasks on tables and (2) a separation of the logical plan for decision of the

cleaning tasks and a physical plan for optimizing the choice of the techniques. The advantage of AJAX compared with other tools is that the separation of the logical and physical levels of data cleaning process enables specifying a series data cleaning tasks using a declarative language and specific optimized algorithms can be selected to implement these data cleaning tasks at the physical levels. For example, considering the matching task for different fields of a database table, the matching of 'personal names' and the matching of 'company names' may be associated with different techniques according to the different physical plans rather than applying a single non exhaustive matching algorithm.

The main contribution of ARKTOS is the presentation of a uniform model covering all the aspects of a data warehouse ETL process. Regarding the types of dirty data that could be dealt with in ARKTOS, data transformations either on schema level or instance level are supported in ARKTOS. Additionally, some integrity constraints enforcement is provided in ARKTOS to prevent primary key violation, reference violation, null value existence and uniqueness violation. Similar to AJAX, these data cleaning tasks can also be specialized with declarative definition languages. Two declarative definition languages are developed in ARKTOS. However, ARKTOS supports a graphical method for a user to specify these cleaning tasks. Compared with AJAX, this is an advance, where a user can compose a scenario with these cleaning tasks and link them in an execution list graphically. Although the authors of AJAX and ARKTOS mentioned the organization of multiple data cleaning tasks in a program, users are required to organise the multiple tasks. In these tools, it is the users who tailor the set of different data cleaning task to be executed according to their individual preferences. Developers of these tools have not undertaken any further investigations on the 'ordering' problem when multiple cleaning tasks are required.

IntelliClean is a knowledge-based framework mainly deals with the problem of

object identification. The detection of duplicate records is totally depends on the rules derived naturally from the business domain. The drawback is that for some complex rules, hand coding is required which decreases the degree of automation. Although the function of merging detected duplicate records is also supported in AJAX, the exclusively developed method to compute the transitive closure during the merging of records increases the recall in IntelliClean. However, compared with Febrl, the algorithms provided in IntelliClean are limited. For example, regarding the algorithms used for the duplicate record detection, only 'SortedIndex' is available in IntelliClean. SortedIndex is a good method to deal with duplicate record detection in large datasets. However when small datasets are involved, the 'FullIndex' clearly is a good solution to improve the effectiveness of the detection. This is a drawback for IntelliClean compared to AJAX or Febrl, in which all these solutions are supported to cope with different situations.

Febrl is an open source data cleaning and record linkage system which includes a variety of techniques for data standardization and duplicate record detection. An advantage of Febrl is the provision of a graphical user interface to its user. Compared with the tools such as AJAX, ARKTOS, IntelliClean, this is especially helpful for users who do not have any programming skills.

Regarding the dirty data types addressed in Febrl, only data standardization and duplicate record detection are supported in Febrl. Additionally, data standardization only supports some domain specific instance-level data transformation. Compared with Potter's Wheel, AJAX, ARKTOS, schema-level data transformation is not supported in Febrl. Besides, Febrl does not support any solution to detect anomalies based on the transformed data as Potter's Wheel does. In Potter's Wheel, as soon as a date value like 'March 1, 20111' is transformed to the expected format '01/03/20111', the sub-component of '20111' in this date value will also be flagged as an outlier with the help of an appropriate algorithm. However, in Febrl, such a

further detection on the transformed data values is not supported. In Febrl, the tasks of data standardization and duplicate record detection have to be specified and executed respectively and can not be performed in a single data cleaning process and this brings in a low efficiency compared with AJAX, IntelliClean which can handle with the multiple data cleaning tasks in a single data cleaning process.

Additionally, unlike AJAX and IntelliClean, Febrl does not support a flexible merging of the linked records into a linked output dataset. In AJAX and IntelliClean, transitive closure calculation and a merging of the linked records are all supported. Without a proper merging function towards the detection results, it is difficult for users to analyze the quality of the detection.

Besides, currently, Febrl only supports three types of text file formats as the input data: CSV, TAB, and COL. Unlike the other tools, loading input data from a database and writing the output data back into a database are not supported in Febrl.

2.4 Data quality, data quality dimensions and other related concepts

A large quantity of data can be created, stored and processed by companies with recent advances in technology. As data increasingly used to support organizational activities such as data warehousing applications, poor quality data may negatively affect organizational effectiveness and efficiency. In this section, data quality, data quality dimensions, the cost and impact of poor data quality as well as data quality assessment are reviewed.

2.4.1 Data Quality

Quality plays an important role as one of the powerful competitive advantages for those companies that run businesses in the information industries. Data quality is regarded as the basis of an information system [8, 25, 48-54].

From the literature, the term ‘data quality’ is complex and still no widely accepted definition exists. For example, from the standpoint of feedback-control systems, data quality is defined as the measure of the agreement between the data views presented by an information system and that same data in the real world [55]. A system’s data quality rating of 100% would indicate, for example, that the data views are in perfect agreement with the real world, whereas a data quality rating of 0% would indicate no agreement at all. Since no serious information system has data quality rating of 100%, the real concern with data quality is to ensure that the data quality system is accurate enough, timely enough, and consistent enough for the organization to survive and make reasonable decisions.

Another approach to define the term ‘quality’, which is widely adopted in most of the quality literature, is focused on the consumer and the product’s fitness for use [56]. The concept of ‘fitness for use’ emphasizes the importance of taking a consumer’s view point of quality because ultimately it is the consumer who will judge whether or not a product is fit for use. However, in order to fully understand the concept, researchers have traditionally identified a number of specific quality dimensions. A dimension or characteristic captures a specific facet of quality. Wang *et al* proposed a framework regarding data quality research. In this work, the authors identified dozens of related research publications with respect to data quality [3]. They found that different combinations of dimensions, as well as a variety of approaches are applied within previous research. The most commonly used dimensions according to their observations are accuracy, timeliness, completeness, and consistency. Some dimensions occurring less frequently are traceability and

credibility. Wang *et al* argue that previous research has mainly focused on the accuracy requirements and since data quality is a multi-facetted concept which includes not only accuracy, more research on other dimensions is needed. Therefore, Wang *et al* drew the analogy between the manufacture of products and the processing of data, i.e., information systems were considered analogous to manufacturing systems, with the difference being that data are used as the raw material, and processed data sometimes referred to as information, are the output.

Adopting a customer perspective similar to the one advocated by Juran [57], Wang *et al* noted that the "use of the term 'data product' emphasizes the fact that the data output has value that is transferred to customers, whether internal or external to the organization". This has become one of the driving forces behind the work by Wang and Strong [4]. Wang and Strong focus on developing a framework that captures the aspects of data quality, which are important to data consumers. In this work, the authors argue that although firms are improving data quality with practical approaches and tools, their efforts tend to focus narrowly on accuracy. A two-stage survey is undertaken in this work. Based on the survey in the first stage, a set of nearly 200 data quality attributes are applied and finally, the authors use factor analysis to narrow the entire set to obtain a much more parsimonious set of 20 dimensions. In the survey of the second stage, the authors reduced this set even further to obtain 15 dimensions. The 15 dimensions are then grouped into four different categories: intrinsic, contextual, representational, and access. The four categories are introduced by the authors as follows: "Intrinsic quality denotes that data have quality in their own right. Contextual quality highlights the requirement that data must be considered within the context of the task at hand. Representational quality and accessibility quality emphasize the importance of the role of systems. These findings are consistent with our understanding that high-quality data should be intrinsically good, contextually appropriate for the task, clearly represented, and accessible to the data consumer." [4].

It is pointed out that the choice of these dimensions is primarily based on intuitive understanding [58], industrial experience [59], or literature review [60]. However, according to Wang *et al*'s work, there is no general agreement on data quality dimensions [3]. Consider the 'accuracy dimension', a dimension which most work has included. Although the term has an intuitive appeal, there is no commonly accepted definition of what exactly 'accuracy' means. For example, Kriebel [60] characterizes accuracy as "the correctness of the output information." Ballou & Pazer [58] describe accuracy as "the recorded value is in conformity with the actual value." Thus, it appears that the term is viewed as equivalent to correctness. However, using one term to define another does not serve the purpose of clearly defining either. In short, despite the frequent use of certain terms to indicate data quality, a rigorously defined set of data quality dimensions does not exist.

Clearly, the notion of data quality depends on the actual use of data. What may be considered good data in one case (for a specific application or user) may not be sufficient in another case. For example, analysis of the financial position of a firm may require data in units of thousands of dollars, whereas auditing requires precision to the cent. This relativity of quality presents a problem. The quality of the data generated by an information system depends on the design of the system. Yet, the actual use of the data is outside of designer's control. Thus, it is important to provide a design-oriented definition of data quality that will reflect the intended use of the information.

2.4.2 Data quality dimensions

From the literature, data quality can be defined as "fitness for use", i.e., the ability of data to meet the user's requirement. The nature of this definition directly implies that the concept of data quality is relative. Some commonly used data quality dimensions include accuracy, completeness, timeliness, and consistency. A dimension captures a

specific facet of quality. Therefore, data quality can be considered as a multi-dimensional concept. These data quality dimensions measure data quality from different angles. To illustrate the multi-dimensional nature of data quality, an example is given below. The following table shows an example of four student records of a university in the UK.

No.	**Name**	**Sex**	**Supervisor**	**R.D**	**G.D**
001	Mark Levison	M	John Smith	2000-10-1	2003-9-1
002	Elizbeth Fraser	F	H.Winston	2001-10-5	NULL
003	Jack Daniel	F	Alex Smith	2002-3-4	2006-9-1
004	Catherine Yang	F	Thomas Lee	2005-4-2	2009-9-21

Table 2.6 An example of four student records of a university in the UK.

In table 2.6, when the "Name" column is checked, a misspelling of a student name is detected, i.e. 'Elizbeth' rather than 'Elizabeth'. With respect to data quality, this problem causes an accuracy problem. Further checking the table, a null value for "G.D" (Graduation Date) is found for Elizabeth. The null value here may have two indications: one is that Elizabeth is still studying in the university and such a graduation date is still unknown. In this case, data quality will not be affected by a null value. Another indication is that Elizabeth has already graduated from the university, but her graduation date has not been stored in the database, in this case, the null value causes a completeness problem as the value of her graduation date is supposed to be there. In the column "Supervisor", suppose it is required that the domain format for the name of the supervisor should follow the pattern of "First Name Last Name". Since "H.Winston" does not conform to this requirement, it will cause an inconsistency problem. This example clearly shows that data quality is a multi-dimensional concept. Wang *et al* discussed how to construct specific data quality dimensions. His group firstly gathered 179 data quality attributes, from the data quality literature, from researchers and from consumers. They used factor

analysis to collapse their list of attributes into fifteen data quality dimensions which is shown in the table below with a brief description for each of data quality dimensions [61].

Data quality dimensions	**Description**
Access Security	Access to data must be restricted, and hence, kept secure.
Accessibility	Data must be available or easily and quickly retrievable.
Accuracy	Data must be correct, reliable, and certified free of error.
Appropriate Amount of Data	The quantity or volume of available data must be appropriate.
Believability	Data must be accepted or regarded as true, real, and credible.
Completeness	Data must be of sufficient breadth, depth, and scope for the task at hand.
Concise Representation	Data must be compactly represented without being overwhelming.
Ease of Understanding	Data must be clear, without ambiguity, and easily comprehended.
Interpretability	Data must be in appropriate language and units, and the data definitions must be clear.
Objectivity	Data must be unbiased (unprejudiced) and impartial.
Relevancy	Data must be applicable and helpful for the task at hand.
Representational Consistency	Data must always be presented in the same format and compatible with previous data.
Reputation	Data must be trusted or highly regarded in terms of their source or content.
Timeliness	The age of the data must be appropriate for the task at hand.
Value-Added	Data must be beneficial and provide advantages from their use.

Table 2.7 Data quality dimensions

From the literature, different researchers have proposed different sets of data quality dimensions. However, due to the contextual nature of quality, there are discrepancies on what constitutes a set of ‘good’ data quality dimensions. Research shows that a

general set of data quality dimensions that can be used to measure the data quality do not exist [4, 62-65].

For example, according to Wang *et al*, the authors argue that "there is no general agreement on data quality dimensions" and three primary types of research (i.e., data quality, information system, accounting and auditing) have attempted to identify appropriate DQ dimensions [66]. The six most important sets of data quality dimensions are presented by Wand and Wang [62], Wang and Strong [4], Redman [63], Jarke [67], Bovee [64], and Naumann [65]. In these six works, six data quality dimensions are considered by the majority of authors: accuracy, completeness, consistency, timeliness, interpretability, and accessibility [68].

With respect to the definitions of each dimension, there is no general agreement on what an appropriate definition is for each data quality dimension. These data quality dimensions are not defined in a measureable and formal way. They have been defined by means of descriptive sentences in which the semantics are consequently disputable. For example, regarding time-related dimensions, Wand and Wang present a 'timeliness' dimension which is defined as "the delay between a change of a real world state and the resulting modification of the information system state" [62]. In Redman's work, a 'currentness' dimension is defined as "the degree to which a datum is up-to-date. A datum value is up-to-date if it is correct in spite of possible discrepancies caused by time related changes to the correct value" [63]. The meanings of these two definitions are quite similar but the names of the two dimensions are different. In Wang and Strong's work, a 'timeliness' dimension is defined as "The extent to which age of the data is appropriate for the task at hand." [4]. A similar definition can be found in Liu's 'timeliness' dimension as "the extent to which data are sufficiently up-to-date for a task." [69]. However, Naumann defines the 'timeliness' dimension as "the average age of the data in a source", which is totally different from Wang and Strong and Liu [65]. Bovee defines the

‘timeliness’ dimension with two levels: ‘currency’ and ‘volatility’ [64]. The currency level of timeliness is defined as “A measure of how old the information is, based on how long ago it was recorded.”, which has the same meaning as the ‘timeliness’ dimension defined by Wang and Strong. The volatility level of timeliness from Bovee is defined as “a measure of information instability-the frequency of change of the value for an entity attribute.”, which corresponds to the ‘volatility’ dimension defined by Jarke [70]. Jarke defines the ‘volatility’ dimension as “the time period for which information is valid in the real world”. This example clearly shows that there is no agreement on the semantics of specific dimensions, i.e., different meanings may be provided by different authors. Besides, there is even no agreement on the names to use for dimensions.

Broadly, the works related with the classification of data quality dimensions can be categorized into two groups: (i) academics’ view of data quality dimensions, and (ii) practitioners’ view of data quality dimensions [71]. Table 2.8 and Table 2.9 present a collection of works under the two groups respectively. In both tables, all dimensions mentioned have been grouped into the four data quality categories proposed by Wang and Strong, namely intrinsic, contextual, representational, and accessibility [4]. Intrinsic quality denotes that data have quality in their own right. Contextual quality highlights the requirement that data must be considered within the context of the task at hand. Representational and accessibility quality emphasize the importance of the role of systems that store and provide access to data.

Category	**Dimension**	**Wang and Strong [4]**	**Zmud [69]**	**Jarke and Vassiliou [70]**	**DeLone and McLean [71]**	**Goodhue [72]**	**Ballou and Pazer [55]**	**Wand and Wang [59]**
Intrinsic	Accuracy	X	X	X	X	X	X	
	Believability	X		X				
	Completeness			X				
	Consistency			X			X	
	Correctness							X
	Credibility			X				
	Factual		X					
	Freedom from Bias				X			
	Objectivity	X						
	Precision				X			
	Reliability				X	X		
	Reputation	X						
	Unambiguous							X
Contextual	Appropriate Amount	X						
	Completeness	X			X		X	X
	Content				X			
	Currency				X	X		
	Importance				X			
	Informativeness				X			
	Level of Detail					X		
	Non-volatility			X				
	Quantity		X					
	Relevance	X		X	X			

	Reliable/Timely		x					
	Source currency			x				
	Sufficiency				x			
	Timeliness	x		x	x		x	
	Usage			x				
	Usefulness				x			
	Value-Added	x						
Representational	Aliases			x				
	Appearance				x			
	Arrangement		x					
	Clarity				x			
	Comparability				x			
	Compatibility					x		
	Conciseness	x			x			
	Consistent	x						
	Format				x			
	Interpretability	x		x				
	Lack of Confusion					x		
	Meaningfulness					x		x
	Origin			x				
	Presentation					x		
	Readability		x		x			
	Reasonable		x					
	Semantics			x				
	Syntax			x				
	Understandability	x			x			
	Uniqueness				x			
	Version control			x				
Accessibility	Accessibility	x		x	x	x		

	Assistance					X		
	Ease of Use	X				X		
	Locatability					X		
	Privileges			X				
	Quantitativeness				X			
	Security	X						
	System availability			X				
	Transaction availability			X				
	Usableness				X			

Table 2.8 Data quality dimensions from academics' view [71]

Category	**Dimension**	**DOD [73]**	**IRI [74]**	**Unitech [75]**	**Diamond Technology Partners [76]**	**HSBC Asset Management [77]**	**AT&T and Redman [78]**	**Vality [79]**
Intrinsic	Accuracy	X	X	X	X		X	
	Completeness	X						
	Consistency	X		X			X	
	Correctness					X		
	Reliability			X				
	Validity	X						
Contextual	Attribute granularity						X	
	Completeness			X		X	X	
	Comprehensiveness						X	
	Currency					X	X	
	Essentialness						X	
	Relevance						X	

	Timeliness	X	X	X				
Representational	Ability to represent null values						X	
	Appropriate representation						X	
	Clarity of definition						X	
	Consistency					X		
	Efficient use of storage						X	
	Format flexibility						X	
	Format precision						X	
	Homogeneity						X	
	Identifiability						X	
	Interpretability						X	
	Metadata characteristics							X
	Minimum unnecessary redundancy						X	
	Naturalness						X	
	Portability						X	
	Precision of domains						X	
	Representation consistency						X	
	Semantic consistency						X	
	Structural consistency						X	
	Uniqueness	X						
Accessibility	Accessibility				X	X		
	Flexibility						X	

	Obtainability						x	
	Privacy			x				
	Reliability (of delivery)		x					
	Robustness						x	
	Security			x				

Table 2.9 Data quality dimensions from practitioners' view[71]

Works from table 2.8 can be further categorized into three groups. The first group is based on an empirical, market research approach of collecting data from information consumers to determine the dimensions of importance to them. Both Wang and Strong [4] and Zmud [72] fall into this group. The second group develops dimensions from the literature. Work by Delaone and Mclean [73], Goodhue [74], and Jarke and Vassilion [75] belongs to this group. They try to cover all possible aspects of data quality by grouping all measures from existing literature. Finally, the third group focus on a few dimensions that could be measured objectively without considering the dimensions importance to data consumers [58, 62]. Table 2.9 presents a collection of work from the practitioners' view. Unlike the academic views, a practitioner's view does not try to focus on covering all possible data quality dimensions but only focus on some specific organizational problems. These practitioners include specialists from organizations, consultants and vendors of products. According to the different contexts involved, different dimensions are defined. Contexts from table 2.9 include: data warehouse development [76, 77], environment with multiple incompatible databases [78], environment in which timely delivery of information is critical [79], and tools for improving the input data quality to databases [80].

A further study with respect to data definition shows that the definition of data is not only a collection of triples <*e,a,v*> where *e* stands for an entity, *a* stands for an

attribute of the entity and *v* is a value selected from the domain of the attribute *a*, but also includes the definition of data representation and data recording [81]. This definition brings the quality of data into three sets of quality issues: the quality of the model or view, the quality of data values themselves, and the quality of data representation and recording [23]. According to David Loshin, "the dimensions associated with data values and data presentation in many cases lend themselves handily to system automation and are the best ones suited for defining rules used for continuous data quality monitoring" [12]. In this research, only data quality dimensions associated with data values are considered. This helps us with generating the proposed rule based taxonomy of dirty data, which will be discussed in detail in chapter 3. Fox *et al* have defined and discussed four dimensions of data most pertinent to the quality of values. The four data quality dimensions are accuracy dimension, completeness dimension, currentness dimension and consistency dimension [23]. These four dimensions are briefly discussed below and they will be used in the proposed dirty data taxonomy in chapter 3.

(i) Accuracy dimension

Suppose a datum is defined as a triple < *e,a,v*> where *e* stands for an entity, *a* stands for an attribute of the entity and *v* is a value selected from the domain of the attribute *a*. The accuracy of the datum refers to the degree of closeness of its value *v* to some value *v'* in the attribute domain considered correct for the entity *e* and attribute *a*. If the datum's value *v* is the same as a correct value *v'*, the datum is said to be accurate or correct. As an example, the value *v* of the attribute "Name" of entity "Student" in table 2.6 (identified by No. 002) is "Elizbeth Fraser" rather than "Elizabeth Fraser". The datum is not said to be correct and causes an accuracy problem. Accuracy problems could be classified as syntactic accuracy problems and semantic accuracy problems respectively. The example of the misspelt name value of "Elizbeth Fraser" belongs to the syntactic accuracy problem. Semantic accuracy problems describe the

case that a data value *v* is itself syntactically correct, but presents a different meaning from *v'*. As an example of semantic accuracy problem, consider a record from table 2.6 again. Suppose in the record with No. 003, if student name "Jack Daniel" is entered in the "Supervisor" field, and "Alex Smith" is entered in the field "Name", then this will cause a semantic accuracy problem, though both name values are syntactically accurate.

(ii) Completeness dimension

Fox *et al* state that completeness is the degree to which a data collection has values for all attributes of all entities that are supposed to have values. The degree of completeness could be measured based on three levels namely tuple, attribute and relation. Tuple completeness measures the percentage of the available values of a record and the total number of attributes of the record. For example, in table 2.6, records with student No. 001, 003 and 004 all have values for each attribute. The tuple completeness for this kind of record is 6/6=1. The record with student No. 002 in this case is 5/6=83.33% since its graduation date is missing. Attribute completeness measures the percentage of non-missing values in a column and the total number of values in such column. As an example of attribute completeness, in Table 2.6, graduation date completeness is 3/4=75%. Tuple completeness measures the percentage of all the non-missing values in the whole table and all the total number of values in such a table. The tuple completeness in table 2.6 is 23/24=95.83%.

(iii) Currentness dimension

Some data in a database are always static. For example, normally a person's birthday, country of birth, skin colour will not change during the whole life of this person. By contrast, some data such as age, address, weight of a person may change as time

goes by. In order to evaluate such temporal data, the currentness dimension is introduced. According to Fox *et al*, a datum is said to be current or up to date at time *t* if it is correct at time *t*. A datum is out of date at time *t* if it is incorrect at *t* but was correct at some moment preceding *t*. As an example of the currentness problem, suppose John Smith had been living in London, UK till the end of 2008. In 2009, he moved to Edinburgh, UK. The residence address for John Smith should also be changed, i.e., in 2009, the time when he moved to Edinburgh, UK, the value of John Smith's residence address should be changed to his address in Edinburgh in the database. If so, the data is said to be current. Due to the late-update of data, currentness problems are observed to cost a fortune. For example, a survey shows that the average annual cost of returned mail is more than $9,000 per company [82].

(iv) Consistency dimension

Data is said to be consistent with respect to a set of data model constraints if it satisfies all the constraints in the set. For example, a database may be designed and maintained independently to serve specific needs. Therefore, the value *v* of the same attribute *a* for the same entity *e* in different databases may be presented in different formats and measured in different units. But when these databases come to be integrated together, inconsistency problems may occur.

2.4.3 Impacts and costs of Data quality

There is strong evidence that data quality problems have become increasingly prevalent in practice with most organizations facing data quality problems [7, 62, 83]. The quality of data is critical to an organization's success. However, not many organizations have taken enough action to deal with data quality problems.

Low quality data brings several negative effects to business users through the loss of customer satisfaction, high running costs, inefficient decision making processes, and

performance [50, 52, 54, 84]. For example, information research has demonstrated that inaccurate and incomplete data may adversely affect the competitive success of an organization [78]. These shortcomings of low quality data affect not only corporate competitiveness but also have negative effects on the organizational culture, such as a demoralization of employees and a trend of mutual distrust within an organization. In a broad spectrum of organizations, a number of business initiatives have been delayed or even cancelled citing poor data quality as the main reason.

Data quality problems can bring significant social and business impacts [25]. For example, because of outdated information in government databases, tax bills continue to be sent to citizens long after their death. Business and industry often have similar data quality problems which are pervasive, costly and disastrous [85-87]. For example, a financial institution is embarrassed due to a wrong data entry of an execution order of 500 million dollars [85]. The explosion of the space shuttle *Columbia* which broke apart during re-entry [88], and the U.S. Navy Cruiser USS *Vincennes* which shot down an Iranian commercial passenger jet with all 290 people killed are all due to the data quality problems [86].

Although more and more references to poor data quality and its impact have appeared in the media, general-readership publications, and technical literature, the necessary awareness of poor data quality, while growing, has not yet been achieved in many enterprises [50].

There are many reasons for the inadequate attention from an organization to data quality, for example, lack of appreciation of the types and extent of dirty data that permeate data warehouses. As practitioners know, creating awareness of a problem and its impact is a critical first step toward resolution of the problem [50]. In this section, the impacts of poor data quality on an organization as well as the costs

associated with data quality problems are reviewed and analyzed.

2.4.3.1 The impact

Poor data quality impacts an organization in many ways, which can be categorized to three different levels [7]:

- Impacts at the operational level: There are three main impacts associated with the operational level namely customer dissatisfaction, increased cost, and lowered employee job satisfaction. With respect to the customers, for example, customers from a telephone company expect their personal information such as their names, postal addresses are correctly stored in the company so that their monthly billing letter or promotion letter will arrive timely. However, problems sometimes happen where customer's information is not correctly addressed either due to a wrongly spelt name or address. Customers sometimes receive their billing letter at a later time or they never receive it and are forced to spend time straightening out their billing errors. Many online shopping customers simply expect the details associated with their order to be correct and they are especially unforgiving of data errors, for example, wrong price tag, wrong status of goods availability. Regarding the cost from the operational level, research shows that cost incurred by customer service organizations to correct customer addresses, orders, and bills will be quite high [89].

- Impact at the tactical level: At the tactical level, an organization's decision making will be compromised due to poor data quality. Since any decision of consequence depends on thousands of pieces of data, defective data will lead to poor decision-making. For example, poor data will make the implementation of data warehouses whose purpose is to help an organization make better decisions, more difficult. The slightest suspicion of poor data quality often hinders managers from reaching any decision. It is clear that decisions based on the

most relevant, complete, accurate and timely data have a better chance of advancing the organizations' goals. At the tactical level, poor data quality will also make it more difficult to reengineer and poor data quality increases the mistrust among the internal organizations.

- Impact at the strategic level: Selecting, developing and evolving a strategy is itself a decision making process. It is clear that strategy making will be adversely affected by poor data quality. It will be a hindrance to develop good strategy without relevant, complete, accurate, and timely data about an organization's customer, competitors, technologies as well as other relevant data. Since strategy has much longer-term consequences to an organization, the impact on this level will be at least as great. When a strategy is rolled out, specific plans are deployed and results are obtained. If the reported results are in some way of poor quality, execution of the strategy will be much more difficult.

2.4.3.2 The cost

From the literature, the costs due to the lack of data quality are substantial in many companies [48, 63, 83, 90]. However, few studies have been done for identifying, categorizing, and measuring the costs associated with low data quality. Most organizations do not have adequate processes and tools to maintain high quality operational data and one of the reasons is due to the lack of appreciation of the knowledge of such costs.

The lack of insight regarding the monetary effect of low quality data, however, is not only an open research problem but also a pressing practitioner issue. For an organization, there are many reasons for the lack of enough attention to data quality problems, for example, lack of knowledge of dirty data. It has been pointed out that calculating the current costs caused by low quality data is difficult because many of these costs are indirect costs which do not have an immediate link between the

inadequate data quality and the negative monetary effects [87].

The term cost in the context of data quality can be defined as a resource sacrificed or forgone to achieve a specific objective or as the monetary effects of certain actions or a lack thereof [87]. From the literature, the cost due to poor data quality for an organization can be broadly categorized into two groups: the cost with low quality data [48, 64, 91-97] and the cost of assuring high quality data [48, 91, 97].

With respect to the cost of low quality data, for example, when customers or citizens' account information are incorrect such as license fees or taxes citizens owe the government, organizations will lose money. When checking customers' information in a database, sometimes it can be found that misspelled customers' names, incomplete postal addresses or outdated address exist. If incorrect customers' postal addresses are used by organizations, clearly, money is wasted when organizations are trying post the marketing materials to their customers.

Furthermore, if such incorrect information is used by organizations for the purpose of analyzing customers' shopping behaviour or customer segmentations, the result obtained will also be incorrect which will result in making an inaccurate strategic and tactical decision and this will further lead to an opportunity loss.

For customers, if an organization repeatedly makes mistakes due to persistent low quality data, customers will feel disappointed and frustrated. They possibly will switch to another competitor for goods and services and the image of the organization will be tarnished.

Incorrect or outdated control data will lead to an invasion of privacy, for example, when database administrators do not properly manage the access control list by not updating it timely. It happens that when some employees have been made redundant,

they can also access or log into the system and obtain some private resources.

Although it rarely happens, low quality data will cause personal injury or even death. For example, wrong instructions due to wrong or outdated data for operating some types of machines such as hazardous equipment will cause accidents and even disasters. Finally, invasion of privacy, personal injury and death as well as significant revenue losses will likely cause lawsuits to organizations. Regarding the cost of assuring data quality, during the process of preventing, detecting, and repairing low quality data, human resources as well as licensing of some software tools are required and will cost organizations. Particularly, manual involvement is typically costly. Table 2.10 and table 2.11 present two cost lists: the cost resulting from low quality data and the cost of assuring data quality.

Costs resulting from low quality data
Higher maintenance costs
Excess labor costs
Higher search costs
Assessment costs
Data re-input costs
Time costs of viewing irrelevant information
Loss of revenue
Cost of losing current customer
Cost of losing potential new customer
‘Loss of orders’ costs
Higher retrieval costs
Higher data administration costs
General waste of money
Costs in terms of lost opportunity
Costs due to tarnished image (or loss of goodwill)

Costs related to invasion of privacy and civil liberties
Costs in terms of personal injury and death of people
Costs because of lawsuits
Process failure costs
Information scrap and rework costs
Lost and missed opportunity costs
Costs due to increased time of delivery
Costs of acceptance testing

Table 2.10 Cost from low quality data

Costs of assuring data quality
Information quality assessment or inspection costs
Information quality process improvement and defect prevention costs
Preventing low quality data
Detecting low quality data
Repairing low quality data
Costs of improving data format
Investment costs of improving data infrastructures
Investment costs of improving data processes
Training costs of improving data quality know-how
Management and administrative costs associated with ensuring data quality

Table 2.11 Cost of assuring data quality

It provides many benefits for an organization to have knowledge of the different costs associated with poor data quality. For example, before investing in a data quality project or initiative, a company may want to examine the potential risks associated with low quality data in order to better position the issue within its corporate context. Instead of an undirected, heuristic search for possible past

experiences or events, direct and indirect data quality costs can be examined in terms of their likelihood and effect, thus contributing to an overall risk assessment of low data quality in an organization.

2.4.4 Data quality assessment

Many research activities have been undertaken and have contributed to improving an organizations' data quality [4, 25, 58, 83, 98-104]. From the literature, the data quality problem has been treated as an important concern in data warehousing projects [8, 59, 105, 106]. However, the ability for an organization to assess its data quality is still weak. Without the ability to assess the quality of their data, organizations cannot assess the status of their organizational data quality and monitor its improvement. For any data quality project, it is important to develop an overall model with an accompanying assessment instrument for measuring data quality. Furthermore, techniques developed to compare the assessment results against benchmarks are necessary for prioritizing the organizations' data quality improvement efforts.

It is well accepted that quality of a product cannot be assessed independent of consumers who choose and use products [107]. Similarly, data quality cannot be assessed independent of the people who use data, i.e., data consumers. Data consumers evaluate data quality relative to their tasks. Data consumers perform many different tasks and the data requirements for these tasks change. It is possible that the same data used by different tasks may require different quality characteristics. For example, it is possible for an incorrect character in a text string to be tolerable in one circumstance but not in another. Therefore, providing high quality data along the dimensions of value and usefulness relative to data consumers' task contexts places a premium on designing flexible systems with data that can be easily aggregated and manipulated [25]. From the literature, data quality is a multi-dimensional concept [4, 58, 62, 63, 83, 100]. In order to have data quality

assessed, both subjective and objective data quality metrics are needed to be considered [50, 108].

Subjective data quality assessment evaluates data quality from views of data collectors, custodians, and data consumers [50] and could adopt a comprehensive set of data quality dimensions which are defined from the perspective of data consumers [4]. The assessment is focussed on the management perspective and concentrates on whether the data is fit for use. During this process, questionnaires, interviews, and surveys can be developed and used to assess these dimensions.

According to Wang *et al*, objective assessments can be task-independent or task-dependent [50]. Task-independent metrics reflect states of the data without the contextual knowledge of the application, and can be applied to any data set, regardless of the tasks at hand. Task dependent metrics, which include the organization's business rules, company and government regulations, and constraints provided by the database administrator, are developed in specific application contexts [50]. During this process, software can be applied to automatically measure data quality according to a set of data quality rules. Dimensions developed from a database perspective can be used for objective assessment [109].

From the literature, information systems have been compared to production systems and an analogy has been proposed between quality issues in a manufacturing environment and those in an information systems environment. In this analogy, data is considered as the raw materials and data products are considered as the output [110, 111]. Table 2.12 shows an analogy between physical products and data products.

Analogy	Input	Process	Output
Product manufacturing	Raw materials	Materials processing	Physical products
Data manufacturing	Raw data	Data processing	Data products

Table 2.12 An analogy between physical products and data products

From table 2.12, three types of data are associated with this analogy. Raw data is considered as raw materials for information manufacturing which are expected to be well structured and stored in the database. Raw data is then composed and transmitted through different business manufacturing processes. Finally, data products are delivered to data consumers for intended use. Therefore, data quality assessment can be carried out with assessment associated with these three types of data, i.e., raw data, component data, and information product.

According to Ge and Helfert [108], objective assessment mainly deals with raw data as well as component data. Subjective assessment deals with the final information products. Within these two types of assessments, data quality dimensions from Wang and Strong are used for the purpose of evaluation [4]. These dimensions are categorized into two different groups, each of which deals with different types of assessment. Figure 2.8 shows the model for the assessment work.

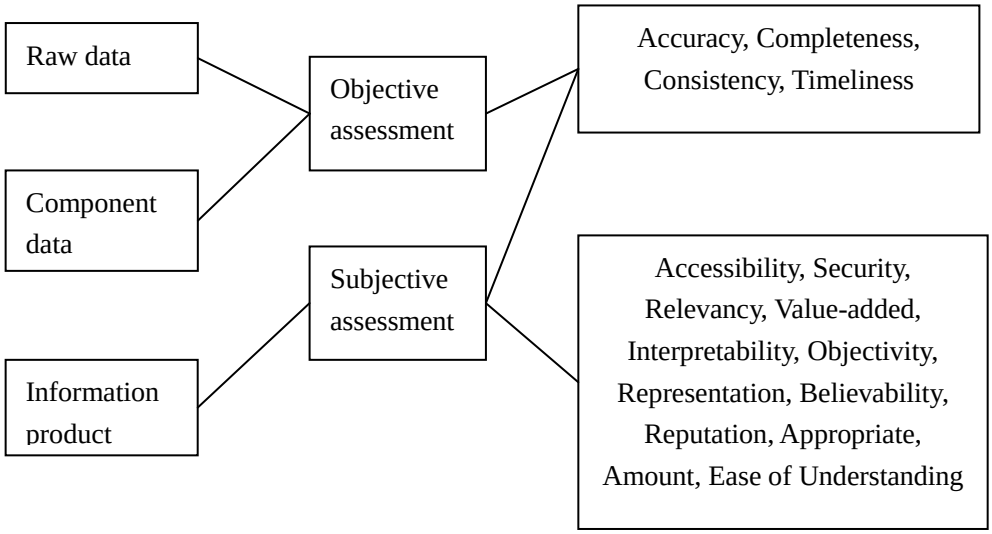


Fig.2.8 A data quality assessment model [108]

In table 2.13, the differences between objective and subjective assessments are listed according to five different aspects: tool, measuring object, criteria, process, assessing results, and data storage.

Feature	**Objective assessment**	**Subjective assessment**
Tool	Software	Survey
Measuring object	Data	Information product
Criteria	Rules, Patterns	Fitness for use
Process	Automated	User involved
Assessing result	Single	Multiple
Data storage	Databases	Business context

Table 2.13 Comparison between objective and subjective assessment [4]

Since subjective criteria and expectations vary from person to person, it is possible that different data consumers will generate different subjective assessment results. Based on the different consumers, subjective assessment results can be positive or negative depending on user requirements [113]. Besides, discrepancies may exist

between the subjective and objective assessments. Based on both assessments' results, we can tell whether the quality of data is high or low. For low quality data, organizations should investigate the root causes and take corrective actions. Regarding the root causes of poor data quality, for a specific context, both data and its environment should be diagnosed carefully. Data environment includes not only database systems but also the related task process mechanisms, rules, methods, actions, policies, and culture that together typify and impact an organization's data quality. From the literature, a group of conditions which will cause poor data quality are identified and analyzed. According to Lee *et al*, these conditions are the commonly ones which are distilled from detailed embedded case studies and content analysis of data quality projects in leading organizations [113]. Table 2.14 lists these conditions.

Condition
Multiple data sources
Subjective judgment in data production
Limited computing resources
Security/accessibility trade-off
Coded data across disciplines
Complex data representations
Volume of data
Input rules too restrictive or bypassed
Changing data needs
Distributed heterogeneous systems

Table 2.14 Root causes of poor data quality

These conditions are summarized in detail below:

- Multiple data sources: Due to the difficulties of ensuring consistent updating of multiple copies of data, inconsistent data values are obtained in multiple data sources for the same information even though they were accurate at a given point of time. Inconsistent data values may also happen due to the different use of measurements, e.g. the different level of units applied in different data sources. However, when a consistent value is required under some special context, the data quality becomes defective. In an organization, this problem happens frequently. There may be multiple systems designed for an organization for different purposes such as financial use, billing use or human resource management use. It may happen that procedures for collecting the same input information vary by different systems in multiple data sources and inconsistencies are observed from multiple sources. This may cause serious problems, e.g., consumers may stop using the information because inconsistencies lead them to question its believability.

- Subjective judgment in data production: Subjective judgment may be involved with information collection and data quality problems may arise due to the biased information produced by subjective judgment. These problems are often hidden from data consumers because the extent to which judgment is involved in creating it is unknown to them. However, it is not proposed that human judgment should be eliminated from information production as some information can only be produced subjectively. Rather, better extended training for data collectors, improvement of the data collectors' knowledge of the business domain, and clear statement and communication about how specific subjective judgments are to be made is encouraged as a solution.

- Limited computing resources: Information may be inaccessible due to the limited computing resources, which may lead to inaccurate and incomplete information. Tasks accomplished without the complete information will lead to poor decision making.

- Security/Accessibility trade-off: Easy access to information may conflict with requirements for security, privacy, and confidentiality. For data consumers, high-quality information must be easily accessible. However, ensuring privacy, confidentiality, and security of information requires barriers to access. Therefore, with respect to high quality data, conflict exits between the accessibility and security dimensions. For example, patients' medical records contain confidential information, yet analysts need access to these records for research studies and management decision making.

- Coded data across disciplines: With technological advances, it is possible to collect and store many types of information, including text and images. Representing this information for easy entry and easy access is an important issue. However, coded data from different professional areas are difficult to decipher and understand. For example, in some hospitals, detailed patients care notes still remain in paper form due to the cost of converting them to electronic form. Deciphering the notes and typing them is time consuming. Some information has to be dictated by the doctor manually.

- Complex data representation: Although advanced algorithms are available for automated dealing with numeric values, they are not available when facing instances of text and image information. With respect to these non-numeric values, data consumers require more than access to them. Functions such as aggregation, manipulation and trend identification are required by consumers for analytical work. This problem is manifested as information that is

technically available to information consumers but is difficult or impossible to analyze.

- Volume of data: Large volumes of stored information make it difficult to access required information in a reasonable time. When dealing with large volumes of data, problems may happen to those responsible for storing and maintaining data as well as those responsible for searching for useful data. For example, customers may expect their telephone company will have an immediate access to their individual billing records in order to resolve their billing questions. However, telephone companies may find it difficult to offer their customers such an immediate service when they have to face large volumes of billing transactions hourly. As another example, when dealing with the duplicate record detection task, in order to achieve a high degree of accuracy, a one-to-one comparison among two records is needed. This will generate a quadratic cost and is not acceptable when the data volume involved is large.

- Input rules too restrictive or bypassed: Input rules are used for imposing necessary controls on data input in order to achieve a high level degree of accuracy. However, as has been pointed out, improving data quality requires attention to more than just accuracy. Other considerations such as usability, usefulness also need to be included. When input rules are too restrictive, data may get lost and produce missing information because they may be unable to fit the field, or erroneous data may be entered into a field due to arbitrarily changing a value to fit such input rules by the data entry clerk. In this case, both accuracy and completeness problems are introduced.

- Changing data needs: Data is only of high quality when they satisfy the needs of data consumers. However, with multiple consumers' special needs, it is difficult to provide high quality data to satisfy all consumers' needs. Besides, when these

needs change over time, the quality of data will also deteriorate even though initially they are good.

- Distributed heterogeneous systems: The most common problem associated with distributed systems is inconsistent data, that is, data with different values or representations across systems. Data with different values may be generated from multiple sources or created by inconsistent updating of multiple copies. Data with different representations becomes a problem when integrating across autonomously designed systems.

As stated by an old aphorism: ‘‘an ounce of prevention is worth a pound of cure.’’ Organizations must not only develop tools and techniques to rectify data deficiencies but also institutionalize processes that would identify and prevent root causes of poor data quality. Awareness will require that organizations quantitatively assess both subjective and objective metrics of data quality.

2.5 Conclusion

High quality of data is a key to today’s business success. Among the many factors caused poor data quality, dirty data existing within data sources is a main reason. In this chapter, four existing research works from the literature associated with identifying dirty data that affect data quality were reviewed, which provides an appreciation of the types and extent of dirty data within data sources. In order to ensure high quality data in an organization, cleaning these dirty data existing in data sources in a proper way is necessary and a data cleaning process which can monitor, analyze and maintain the quality of data is highly recommended. From the literature, many data cleaning techniques and approaches exist to facilitate a data cleaning process. A group of selected data cleaning techniques and approaches are reviewed and analyzed in this chapter. Especially, the critical analyses regarding the

advantages and disadvantages of these approaches provide valuable information regarding the design of the proposed data cleaning framework in Chapter 4. Data cleaning tools and frameworks are crucial for making the data cleaning techniques and methodologies effective. To summarise, there are still some challenges regarding the design of a data cleaning approach. To address these challenges, the following considerations are presented:

(i) An analysis among the five data cleaning approaches shows that, the two frequently addressed cleaning tasks are (1) instance-level data standardization and transformation and (2) duplicate records elimination. Some approach only focus on dealing with one of these two tasks exclusively. Although from the literature, some work has been done for the purpose of generating a taxonomy of dirty data [6, 30], according to the knowledge of the author, there is no such a data cleaning tool that can deal with all the dirty data types mentioned from these works. In practice, cleaning all dirty data types introduced by the two taxonomies mentioned above is unrealistic and simply not cost-effective when taking into account the needs of a business enterprise. This problem in this thesis is defined as DDS problem. Thus, the power of a selection of different dirty data types to deal with under different situations is expected for a data cleaning approach.

(ii) According to Galhardas *et al*, the more dirty data involved, the more difficult to automate their cleaning within a fix set of transformations [18]. Currently, in existing data cleaning tools, organizing the multiple cleaning tasks in a proper cleaning sequence is not supported and is totally depends on a user's preference.

This brings two drawbacks: the first drawback is that the human involvement during a data cleaning process may bring down the degree of automation when performing data cleaning tasks. Ideally, the process of detecting and correcting the dirty data should be performed automatically. However, it is known that fully automatically

performing data cleaning is nearly impossible in most of cases especially when exceptions happen during the cleaning process and an expert is required to make a judgement. Therefore, declarative, semi-automatic approaches are feasible and acceptable for developing a data cleaning approach.

Considering the semi-automatic approach, the idea of dividing the data cleaning process into several sub-processes which separate the sub-processes that can be executed fully automatically from others is a good solution [32]. But still, the executions of these sub-processes are needed to be specified in an order. So the second drawback is that, for users who have no knowledge in data cleaning, ordering these sub-processes is difficult and a poor ordering sequence will bring side effects to the final cleaning result as is shown later. Therefore, a semi-automatic data cleaning approach with the power of automatically ordering the associating the related data cleaning tasks is a challenge.

(iii) To develop an effective data cleaning tool, it is necessary that a tool should include various appropriate methods or techniques to deal with a specific data quality problem when different domains are involved. A specific optimized algorithm which is already parameterized is not able to cope with all situations. Choosing a method or an algorithm from a set of alternative algorithms has proven to be a difficult task. It depends on several factors, such as the problem domain and the nature of the errors. Therefore, data cleaning methods/algorithms should be critically analyzed and evaluated based on carefully designed experiments.

According to the studies of the five data cleaning approaches in section 2.2, algorithm selection and algorithm parameter setting depends on user’s preference. This leaves the data cleaning process with two drawbacks: the first is with degree of automation for a data cleaning approach. For example, in Febrl, 26 different algorithms are provided to its users. In order to perform a matching task with Febrl,

the user has to choose one particular algorithm out of these 26 algorithms and the corresponding parameters must also be specified by the user. According to the author's experience of using Febrl, nearly 20% of the total data cleaning time is spent on algorithm selection and parameter setting.

The second drawback is associated with the effectiveness of the data cleaning task. As is mentioned, several factors such as the problem domain and the nature of errors are involved with the selection of a suitable algorithm. As will be shown later, the experimental results in Chapter 5 confirm that the effectiveness and efficiency of a data cleaning task may vary with selection of a different algorithm. For users who have not enough knowledge and experience, an inappropriate selection of algorithms will generate poor cleaning results. Therefore, another challenge for a data cleaning approach is that not only should it include enough techniques for user to choose but it can intelligently help its users to make a choice out of many alternatives when necessary. These considerations will be included during the design of the proposed data cleaning framework.

Finally, the review work regarding data quality and data quality dimensions in section 2.4 provides a solid foundation in designing the proposed rule based taxonomy of dirty data, which is presented in the next chapter.

CHAPTER 3 A RULE-BASED TAXONOMY OF DIRTY DATA

In Chapter 2, literature concerning dirty data type classifications or taxonomies was reviewed. Regarding the dirty data type classifications, some work has been undertaken exclusively to identify problems (dirty data types) that affect data quality and has resulted in taxonomies of dirty data. For example, Kim *et al* [6] and Oliveira *et al* [30] have proposed two different taxonomies of dirty data and have presented 33 and 35 dirty data types respectively.

Some work, although not undertaken exclusively for the purpose of generating a taxonomy of dirty data, has highlighted the problems arising due to poor data quality and groups of dirty data types have been proposed. For example, according to the constraints of Müller and Freytag's pre-defined data model [27], data from data collection that does not conform to the constraints of the data model are considered to be data anomalies. Müller and Freytag roughly classify data anomalies into three different sets, namely syntactical anomalies, semantic anomalies and coverage anomalies and together 8 dirty data types are identified. Rahm and Do [28] distinguish the observed data quality problems into two sets, namely single-source problems and multi-source problems. Within each set, data quality problems have been classified into schema-level problems and instance-level problems respectively. These problems reflect the different dirty data types that could be captured according to different levels and 19 problems have been introduced in their work.

Compared with Müller and Freytag's and Rahm and Do's work, the two taxonomies of dirty data provide many more types of dirty data. Data cleaning is a labour-intensive, time-consuming and an expensive process. In practice, cleaning all dirty data types introduced by the two taxonomies mentioned above is unrealistic and simply not cost-effective when taking into account the needs of a business enterprise. For example, a company might only be able to afford to clean a specific

group of types of dirty data to satisfy some specific needs. The problem then becomes how the business can make a selection according to their different business needs. This problem, mentioned in Chapter 1 is referred to as the Dirty Data Selection (DDS) problem.

Although there are several taxonomies of dirty data existing in the literature, none of them are designed for this purpose. For example, in Oliveira *et al*'s taxonomy of data quality problems, 35 dirty data types have been introduced, which is considered as the most comprehensive taxonomy so far in the literature.

In this case, by only showing these 35 dirty data types, it is difficult to tell which possible dirty data types should be selected to deal with for different data sets. In this chapter, a rule-based taxonomy of dirty data is presented. As is mentioned in chapter 2, dirty data is defined as the data flaws that break any of the pre-defined data quality rules. The taxonomy presents a clear mapping between the data quality rules and dirty data types, which not only covers a larger range of dirty data types than any of the existing taxonomies but can also help dealing with the DDS problem when specific business needs are considered.

3.1 Data quality rules

According to Chanana and Koronios, most data quality problems are not simple violations of declared database integrity constraints, but a large number of real-life data problems are caused by data violating complex underlying business rules or data quality rules often leading to poor data quality [114].

In the proposed context, data quality rules define the business logic of an enterprise and are therefore an underlying reality in an enterprise [115]. Data quality rules are used as descriptive means for encapsulating operational business flows, govern and

guide the way in which an enterprise conducts itself and comply with legal and other regulations. They are defined and owned by business professionals, not IT professionals [116] and they do not contain any control flow statements, which are independent of any implementation techniques.

In the past, data quality rules have been embedded in the system code rather than formalized and articulated separately in simple natural language. With advances in the scale of business, changing business environment, operations at different locations and increased interaction with stakeholders, the business process is now more complex and it becomes difficult and unmanageable to operate the business effectively and efficiently without formalizing these quality rules. As business practices and/or policies change frequently, it becomes very difficult to reflect these changes in the applications implementing them. Rules that are buried in information systems are neither flexible nor easy to modify or change and as a result do not render the business with complete control over its environment [114].

According to David Loshin [12], by relating business impacts to data quality rules, an organization can employ the data quality rules for measuring the business expectations and the improvement of data quality can be viewed as a function of conformance to business expectations. By integrating control processes based on data quality rules, business users are able to determine how best the data can be used to meet their own business needs. Thus, data quality rules play an important role in the improvement of data quality for a business.

In this thesis, dirty data is defined as the data flaws that break any of the pre-defined data quality rules when data quality rules are obtained. Data can be assessed as whether or not the data is dirty according to the description of these rules. This provides agility in responding to the ever changing demands of the business environment. Since the validity of a data value is defined within the context in

which data values appear, one must specifically describe what defines a valid value in order to improve data quality. This is performed by measuring if the values conform to the matching data quality rules.

The approach of cleaning dirty data according to the different data quality rules helps with the separation of business logic from implementation logic and thus provides a solution to respond to the different demands in different business environments. Additionally, the DDS problem introduced in Chapter 1 can be solved well, since it is reasonable for a business enterprise to deal with a few of the most important groups of data quality rules rather than all of the rules, according to its own business priorities. Only dealing with the dirty data reflected in the selected data quality rules helps an organization with reducing the cost associated with the expensive data cleaning tasks, especially when available resources for an organization to perform data cleaning is limited.

From the literature, Chanana and Koronios proposed a set of data quality rules and categorized them into five groups. Table 3.1 shows the data quality rules from Chanana and Koronios' work.

Rule Class	Rule types	Description
C.1 Definitions of reference data	C.1.1 Null values rules	Allows traditional null values like system null, blank, empty fields
		Non-null specifies which null values are not allowed
	C.1.2 Domain membership rules	Enumerated defines a list of valid values
		Descriptive domain uses syntax to establish domain membership
C.2 Mappings between domains rule	C.2.1 Functional domain mapping rules	List of functions describing mapping
	C.2.2 Domain mapping enumeration rules	Specifies those value pairs that belong to the mapping
	C.2.3 Mapping membership rules	Two attribute values must confirm to the mapping
C.3 Value constraints	C.3.1 Value constraints rules	Specifies set of valid values that can be assigned
	C.3.2 Attribute value restriction rules	Data type like integer or string
C.4 Relation rules	C.4.1 Consistency rules	Maintains relationship between two attributes based on actual values of attributes
	C.4.2 Completeness rules	Specifies attribute values on satisfying some condition
	C.4.3 Exemption rules	On meeting a condition, some attributes can have null values
C.5 Cross-table rules	C.5.1 Primary key assertion rules	Attribute belonging to primary key can't have null values
	C.5.2 Foreign key assertion rule	Specifies consistency relationship between tables
	C.5.3 Functional dependency rules	Specify inter-record constraints on records

Table 3.1 Data quality rules

Adelman *et al* also propose a set of data quality rules which, according to the authors, have been categorized into four groups namely: business entity rules; business attribute rules; data dependency rules; and data validity rules. Business entity rules specify rules about business objects or business entities. Business attribute rules are rules about data elements or business attributes. Data dependency

rules specify different types of dependencies between business entities or business attributes. Data validity rules govern the quality of data values [89]. Table 3.2 lists the entire data quality rules based on the four different categories proposed by Adelman *et al*. All tables from the appendix B (B.1~B.4) show all further classified distinct sub rules in detail and each sub rule has been associated with a rule number.

Rule Category	**Data Quality Rule**
1.Business entity rules	R1.1 Entity uniqueness rules
	R1.2 Entity cardinality rules
	R1.3 Entity optionality rules
2.Business attribute rules	R2.1 Data inheritance rules
	R2.2 Data domains rules
3.Data dependency rules	R3.1 Entity-relationship rules
	R3.2 Attribute dependency rules
4.Data validity rules	R4.1 Data completeness rules
	R4.2 Data correctness rules
	R4.3 Data accuracy rules
	R4.4 Data precision rules
	R4.5 Data uniqueness rules
	R4.6 Data consistency rules

Table 3.2 Data quality rules from Adelman et al's work

All data quality rules from Chanana and Koronios and Adelman et al are compared and the result is shown in Table 3.3.

Chanana and Koronio's work	**Adelman *et al*'s work**
C.1.1 Null values rules	R 4.1.4
C.1.2 Domain membership rules	R2.2.1, R2.2.5
C.2.1 Functional domain mapping rules	R3.2.2
C.2.2 Domain mapping enumeration rules	R2.2.1
C.2.3 Mapping membership rules	R3.2.3
C.3.1 Value constraints rules	R2.2.1
C.3.2 Attribute value restriction rules	R2.2.3
C.4.1 Consistency rules	R4.6.1
C.4.2 Completeness rules	R4.1.4
C.4.3 Exemption rules	R4.1.4
C.5.1 Primary key assertion rules	R1.1.2
C.5.2 Foreign key assertion rule	R4.1.2
C.5.3 Functional dependency rules	R3.1.1

Table 3.3 A comparison

The comparison result between the two works shows that Adelman *et al* provide an even larger comprehensive collection of data quality rules. All data quality rules mentioned by Chanana and Koronios's work can also be found in Adelman *et al*'s work. Therefore, Adelman *et al*'s collections of data quality rules are used in the proposed research work. We use these four groups of data quality rules from Adelman *et al* to classify dirty data types into four different categories. According to Adelman *et al*, the four groups of data quality rules are further divided into a list of sub-rules from which a tree structure classification of data quality rules is obtained. By analyzing data quality rules on the leaf nodes, we have identified the dirty data types in each category.

3.2 Dirty data types

Four groups of dirty data types are obtained according to the four different rule categories from table 3.2. Each group of dirty data types is detailed below.

(i) *Business entity rules related dirty data types:*

Business entity rules specify rules about business entities which are subject to three data quality rules namely entity uniqueness rules, entity cardinality rules and entity optionality rules. Within this group, the following dirty data types are identified:

- Cardinality relationship problem: Cardinality refers to the degree of a relationship, i.e., the number of times one business entity can be related to another. As an example of this problem, the number of employees by counting the number of tuples from the Employee table, is not the same as the number of employees by summing the number of employees in each department in the Department table.
- Recursive relationship problem: A recursive relationship corresponds to cycle situations among two or more related tuples in a self or reflexive relationship. As an example of this problem, suppose in a department of a university, one person may supervise many other persons and each supervised person may have many supervisors at the same time. Such information is recorded in the table *people (ID*, name, supervise)*. Suppose the information 'Jack is supervising Rose and Rose is supervising Jack' is found in the table. Clearly, this is not going to happen in the real world.
- Optionality relationship problem: the entity optionality rule identifies the minimum number of times two business entities can be related. For example, an online store requires that when a customer has purchased a product on line,

the customer's delivery information must be in the delivery table. Otherwise, a missing tuple from the delivery table will cause a problem such as an undelivered item.

- Reference defined but not found: When a relationship is instantiated through a foreign key, the referenced instance of the entity must exist in the related table.

(ii) Business attribute rules related dirty data types:

Business attribute rules specify rules about business attributes or data elements, which are subject to two data quality rules namely data inheritance rules and data domain rules. As data inheritance rules are object oriented related rules, we do not consider this rule in our work because we consider only database applications. Therefore, in this group, the following dirty data types are identified:

- Set violation: For an enumerated data type, its value should be within the allowable value set. For example, suppose the allowable data value set for "city" attribute is (London, Edinburgh, Manchester, Birmingham), then the value of "New York" is not allowable.
- Data value out of value range: As an example of this problem, suppose the age of human being in a database is defined as "18<=age<30". It is not allowed that an age value of '10' or '35' is entered in the table.
- Data value constraint violation: When some constraints are used to regulate data values, the data value should conform to those constraints. A constraint may be used to regulate a single piece of data or multiple data values. For example, a medical experiment requires the age of the people who participate should be below 30 (inclusive). Then the constraint for "age" attribute is "age<=30". If data has been found that its age value is "35", then such data is not expected in the table.

- Use of wrong data type: When the value of an attribute such as “Name” is set to be a string data type, it is not expected that a numeric value be found for the “Name” attribute.
- Syntax violation: Syntax violation happens when data value does not conform to the defined pattern or format for its attribute. For example, when the format of “Date” attribute is defined as the pattern of “DD/MM/YYYY”, then the value of “2010-03-05” is not expected. The correct value should be “05/03/2010”.

(iii) Data dependency rules related dirty data types:

Data dependency rules apply to data relationships between two or more business entities or business attributes. The dirty data types identified in this group are:

- Data relationship constraint violation: As an example of this problem, an employee who has been assigned a project is not allowed to enroll in a training program, i.e., this employee’s data is not supposed to be found in the training table.
- Contradiction data: The existence of an attribute value is determined or constrained by the value of another attribute. For example, suppose it is defined that when the status of a loan is “funded”, then the value of loan amount must be greater than zero.
- Wrong derived field data: This problem occurs when a data value is derived from two or more other attribute values. For example, a miscalculation of an employee’s income by miscomputing the tax will result in a wrong derived field data.
- Wrong data among related attributes: This problem occurs when the value of one attribute is constrained by the value of one or more attributes in the same

business entity or in a different but related business entity. For example, the value of annual expenses in a department is constrained by the sum of all distinct expenses in that department.

(iv) Data validity rules related dirty data types:

Data validity rules govern the quality of data values, there are six data validity rules (Rule 4.1~ Rule 4.6, see table 3.6). The dirty data types identified by the six validity rules are:

- Missing tuple: Entity completeness requires that all instances exist for all business entities, i.e., all records are present in the table.
- Missing value: It is required that all attributes for a business entity contains all allowable values. It should be clear that Null value is different from missing value. When a constraint of "null-value allowed" is enforced on the data set, null value indicates "value unknown or nonexistent". A missing value simply indicates whether a value should exist for the attribute or not.
- Meaningless data value. The data value for an attribute must be correct and reflect the attribute's intended meaning. When the data value is beyond the context of the attribute, the data value is a meaningless data value. For example, the value for the attribute "address" is defined as a set of allowable characters which reflect a person's address in the real world. If "£$%S134" is entered, it does not make any sense as valid address data.
- Extraneous data entry: An example of extraneous data entry is the entry of address and name in a name field.
- Lack of data elements: An example of this problem is when a part of post code is missing from attribute "PostCode" , i.e., "5DT" missing from "EH10 5DT" .

- Erroneous entry: An example of erroneous entry is when a student's age is entered as "26" rather than the student's real age "27".
- Entry into wrong field: This problem occurs for example when the value of a person's name is entered into its address field.
- Identity rule violation: As an example of this problem, suppose in table *employee (Emp_No., Name, Emp_NIN, DoB)*, Emp_No. is defined as the primary key. According to the values of Emp_No. from employee table, the uniqueness of Emp_No. is guaranteed. But it does not mean that each employee is properly identified in the data. For example, a person may have two records with two distinct Emp_No. but identical values for national insurance number (NIN). Suppose it is required that each person can only has one unique Emp_No in the table. Obviously, they are duplicate records referring to the same person.
- Wrong reference: This is the case when a reference is defined but its value is wrong which breaks the attribute's dependency rules.
- Outdated value: It is required that the data value must be accurate in terms of its state in the real world. If not, its value is said to be an outdated value because it does not represent its real state in the real world.
- Imprecision: It is required that all data values for a business attribute must be as precise as required by the attribute's business requirements. As an example of imprecision data, suppose an analysis of the financial position of an auditor requires the value of the data has precision to the pence, if the value is based on the unit of pounds, then the data is imprecise.
- Ambiguous data: The use of abbreviation of data for instance, sometimes may cause an ambiguous meaning which is not as precise as required by the attribute's intended meaning. For example, when an abbreviation word "MS" is used to represent a company's name, it is difficult to tell whether it stands for "Morgan Stanley" (a global financial services firm) or "Microsoft" (a

global software company) when both of the companies have been recorded in the same data source.

- Misspelling: A misspelling problem, for example, when “John Smith” is entered as “Jonh Smyth”.
- Duplicate record in single/multi data source: Rule 4.5 specifies that each business entity instance must be unique. Duplicate records may happen for example, when a person’s name and address are represented in different ways, the same entity may be represented more than once in the same or different data sources.
- Inconsistent record in single/multi data source: Rule 4.6 specifies the data value should be consistent. Inconsistent data can be found in both single and multi-sources. For example, in different data sources, the data value of the same person’s address may be recorded differently. Suppose this person has only one valid address, these records are inconsistent records.
- Different representations for the same data: in addition to inconsistent record, data conflicts may arise when multiple data sources are integrated. Usually, different data sources are typically developed and maintained independently to serve specific needs. When these data sources are integrated, due to the different representations for the same data, problems are observed. Specifically, these differences may be due to the different use of abbreviations, special characters, word sequence, measurement unit, encoding format, aggregation levels and alias names.

According to the descriptions of the data validity rules, some schema-level problems can also be identified. For example, one of the data completeness rules requires that all business attributes for each business entity exist. In this case for example, if an employee’s address is represented in a different number of fields in different data sources and they are each correct in their own data source, when they come to be

integrated, problems will occur. In data uniqueness rules, two of them are related with the definition of attributes (homonyms and synonyms) which are also related to schema-level problems. As we do not consider schema-level problems in our research, dirty data related with the schema-level will not be considered in the proposed taxonomy. With the above dirty data types analyzed based on data quality rules, table 3.4 lists these dirty data types, each of which has been assigned a type number (DT.1 ~ DT.38).

No.	**Dirty Data Type**
DT.1	Cardinality relationship problem
DT.2	Recursive relationship problem
DT.3	Optionality relationship problem
DT.4	Reference defined but not found
DT.5	Set violation
DT.6	Data value out of value range
DT.7	Data value constraint violation
DT.8	Use of wrong data type
DT.9	Syntax violation
DT.10	Data relationship constraint violation
DT.11	Contradiction data
DT.12	Wrong derived field data
DT.13	Wrong data among the related attribute
DT.14	Missing tuple
DT.15	Missing value
DT.16	Meaningless data value
DT.17	Extraneous data entry
DT.18	Lack of data elements
DT.19	Erroneous entry

No.	Dirty Data Type
DT.20	Entry into wrong field
DT.21	Identity rule violation
DT.22	Wrong reference
DT.23	Outdated value
DT.24	Outdated reference
DT.25	Imprecision
DT.26	Ambiguous data
DT.27	Misspelling
DT.28	Duplicate record in single data source
DT.29	Duplicate record in multi data source
DT.30	Inconsistent record in single data source
DT.31	Inconsistent record in multi data source
DT.32	Different representations due to abbreviation
DT.33	Different representations due to special characters
DT.34	Different representations due to word sequence
DT.35	Different representations due to measurement unit
DT.36	Different representations due to encoding format
DT.37	Different representations due to aggregation level
DT.38	Different representations due to use of alia name

Table 3.4 Dirty data types

3.3 The taxonomy

In Table 3.2, data quality rules have been organized in a tree structure. The proposed taxonomy follows the same structure and classifies the dirty data according to the four different categories of data quality rules. As the 38 dirty data types are obtained based on analyzing the rules on the leaf nodes, the four categories of dirty data have

been further classified into distinct dirty data types according to the corresponding rules on the leaf nodes. Table 3.5 shows the proposed taxonomy.

Dirty Data Category	**Data Quality Rules**	**Dirty Data Type**
Business entity rules related dirty data	R1.2 Entity cardinality rules	DT.1, DT.2
	R1.3 Entity optionality rules	DT.3, DT.4
Business attribute rules related dirty data	R2.2 Data domain rules	DT.5~DT.9
Data dependency rules related dirty data	R3.1 Entity relationship dependency rules	DT.10
	R3.2 Attribute dependency rules	DT.11~DT.13
Data validity rules related dirty data	R4.1 Data completeness rules	DT.14, DT.15
	R4.2 Data correctness rules	DT.16~DT.20
	R4.3 Data accuracy rules	DT.21~DT.24
	R4.4 Data precision rules	DT.25~DT.27
	R4.5 Data uniqueness rules	DT.28, DT.29
	R4.6 Data consistency rules	DT.30~DT.38

Table 3.5 Rule-based taxonomy of dirty data

In this taxonomy, 38 different dirty data types have been identified under different data quality rules, which forms an even larger collection of dirty data compared with any of the existing taxonomies or classifications [6, 27, 28, 30]. In the category of business attribute rules, 5 dirty data types are identified. There are 4 dirty data types identified with each of the categories of business entity rules and data dependency rules. The majority of dirty data types are related to the category of data validity rules, which has 25 dirty data types. This is because the data value related problems are much more common than others. In total, there are 38 distinct dirty data types identified. The proposed taxonomy has considered dirty data types not only appearing within both a single data source and multiple data sources, but also from the angles of both a single relation and multiple relations. Compared with the four existing works [6, 27, 28, 30], it is clear that the proposed taxonomy is most complete. For example, D26, D38, D12, D24, D13, D10 are the dirty data types that are not mentioned by Müller and Freytag [27] and Rahm and Do [28]. Compared with the two formal taxonomies by Kim *et al* [6] and Oliveira *et al* [30], the proposed taxonomy not only covers all instance level dirty data types from these two taxonomies but also includes a new dirty data type (D.18, lack of data element). However, due to the research scope, schema-level related problems are not considered in the proposed taxonomy. For example, naming conflicts and structure conflicts are two schema level heterogeneities mentioned by Rahm and Do [28]. Similarly, two schema-level problems are also identified by Oliveira *et al* [30] (i.e., Syntax inconsistency both in multiple relations in a single data source and among multiple data sources). This consideration agrees with the suggestion made by Kim *et al* [6]. A systematic classification of schema related problems has been proposed by Kim and Seo [117], which covers all the schema-related problems mentioned in the two existing works [27, 28]. Although it is believed that this taxonomy is very comprehensive, still, this does not ensure that it covers all possible dirty data types that may exist. However it is believed that most usual or unusual dirty data types are covered in the proposed taxonomy.

As mentioned in Chapter 1, in practice, cleaning all dirty data types introduced by any of existing taxonomies is unrealistic and not cost-effective when taking into account the needs of a business enterprise. The proposed rule-based taxonomy presents a special structure to organize the different types of dirty data according to the different quality rules. This structure will help with providing a solution to respond to the different demands in different business environment. Only dealing with the dirty data reflected in the selected data quality rules helps with reducing the cost associated with the expensive data cleaning tasks and solves the proposed DDS problem. A method to deal with the proposed DDS problem will be further detailed in the next Chapter.

3.4 Conclusion

In this chapter, the proposed rule-based taxonomy of dirty data is presented. Compared with existing works, this taxonomy includes 38 distinct dirty data types and provides a larger collection of dirty data types than any of existing taxonomies.

Associating dirty data with data quality rules will provide several benefits. For example, it provides agility in responding to the different demands from different business environments. This enables the separation of business logic from logic implementation and people who try to evaluate and improve the data quality of an organization will only focus on the data quality rules without considering the actual techniques regarding dirty data cleaning. On the other hand, developers who try to develop techniques to cope with different dirty data types will not be distracted by the changing of different business environments. Additionally, since it is reasonable for a business enterprise to pick up a few of the most important groups of data quality rules rather than focusing on all rules according to its own business priorities,

dirty data associated with the selected data quality rules will be firstly dealt with. In this way, the proposed DDS problem is solved by only dealing with the dirty data reflected in these selected data quality rules.

Although some existing work has also proposed a large collection of dirty data types such as the collection of 35 dirty data types by Oliveira *et al*, by only looking at these dirty data types, it is difficult to tell which group of dirty data should be firstly considered and it would be very expensive for the system to run all algorithms for all the possible dirty data candidates which is exactly the DDS problem. With the help of the proposed rule based taxonomy of dirty data, a method to deal with the DDS problem could be developed exclusively to be used by business enterprises to solve the DDS problem, by prioritizing the expensive process of data cleaning, therefore maximally benefitting their organizations. In next chapter, this method will be detailed in the proposed data cleaning framework.

CHAPTER 4 A DATA CLEANING FRAMEWORK

4.1 Introduction

High quality data plays an important role in the success of data base applications. Data cleaning is a way to maintain high quality data and is one of the crucial tasks to improve the efficiency of building up the database applications such as a data warehouse (DW). Research shows that nearly half the time of dealing with back-end issues such as readying the data and transporting the data to a DW can be attributed to the activities associated with data cleaning [32]. Regarding the data cleaning process, two considerations need to be addressed: (i) how to reduce the time cost during the data cleaning process, i.e., the improvement of the efficiency of data cleaning process, (ii) how to improve the degree of automation during the data cleaning process.

Recall the data cleaning process presented in Fig.1.1, ideally, the detection and correction of error instances are expected to be performed automatically. However, from the literature, a fully automatic data cleaning tool does not exist. In most cases, it is impossible to have the data cleaning process executed fully automatically with current existing data cleaning approaches. There are many factors which need to be considered during the data cleaning process such as the problem domain, the various dirty data types involved, and sometimes, human involvement is required during the data cleaning process. For example, according to Müller and Freytag, "the process of data cleaning cannot be performed without the involvement of a domain expert, because the detection and correction of anomalies requires detailed domain knowledge" [27].

However, due to the large amount of data that are usually involved, data cleaning

should be as automatic as possible [27]. Therefore, declarative, semi-automatic approaches are feasible and acceptable for developing data cleaning tools [32]. In Chapter 2, data cleaning approaches from the literature were reviewed. There are two main data cleaning activities addressed in these tools: (1) data standardization and transformation and (2) duplicate record elimination. Regarding the 38 dirty data types from the proposed taxonomy of dirty data, the existing data cleaning approaches have only addressed a small number of these 38 dirty data types. For other dirty data types that can not be cleaned with existing data cleaning approaches, users have to seek other solutions to deal with them exclusively. This however requires much user effort during the data cleaning process. Therefore, an ideal data cleaning approach should be able to provide as many solutions for the various types of dirty data as possible.

Regarding the degree of automation during a data cleaning process, frequent human involvement is not encouraged though it is unavoidable. During the data cleaning process, human involvements should be reduced as much as possible and leave most of the data cleaning activities to be handled by the tool. In the data cleaning approaches studied, human involvement is required during the data cleaning process. This is especially required in the following three cases: (1) select a suitable algorithm and set the necessary parameters of the selected algorithm, (2) organize a sequence to perform the multiple data cleaning activities involved in the data cleaning process, and (3) deal with exceptions.

With respect to the first case, for each type of dirty data involved during the data cleaning process, an appropriate method must be firstly selected and then applied. Choosing such a method has proven to be a difficult task as it depends on several factors such as the problem domain and the nature of dirty data types [31]. Currently, existing data cleaning approaches either adopt one fixed method to clean dirty data without considering the different problem domains or they require users to select a

method from a list of alternatives. For example, Febrl provides multiple solutions to deal with the problem of duplicate record detection and users have to choose one solution out of the many alternatives in order to have the duplicate records detected. As mentioned in Chapter 2, Febrl does not supply any recommendations or help with selecting an appropriate method to cope with different problem domains. AJAX, on the contrary, will provide an optimal solution in its physical level according to the information provided by the user on the logical level. This is an advance compared to Febrl regarding the degree of automation. However, AJAX does not support as many techniques as Febrl does. This is a drawback regarding the ability to cope with different problem domains.

With respect to the second case, when multiple data cleaning activities are involved, the organization of a sequence to execute those associated algorithms is usually determined by a user rather than the system. For example, in ARKTOS, the user can customize multiple cleaning tasks either graphically or declaratively. Users of ARKTOS are responsible for specifying the correct order to execute these tasks. In Febrl, a data cleaning task is performed individually. For example, data standardization and duplicate record detection need to be executed separately.

In order to develop an effective and efficient data cleaning approach, it is necessary to allow users to select an appropriate method for different problem domains and it should provide a mechanism for users to organize an appropriate order regarding the multiple data cleaning activities. Regarding the organization of the multiple cleaning activities during the data cleaning process, Müller and Freytag proposed a sequence associated with different data cleaning activities as: format adaptation for tuples and values → integrity constraint enforcement → derivation of missing values from existing ones → removing contradictions within or between tuples → merging and eliminating duplicates → detection of outliers [27]. However, there is no mention in their work whether there is any particular reason to perform these operations in this

order. As will be seen later, ordering the multiple activities in a data cleaning process is a complex task where multiple factors such as the problem domain, the nature of the selected algorithm must be considered. A different order may result in different data cleaning performance during the data cleaning process.

In this chapter, a data cleaning framework is proposed which aims to challenge the following issues: (1) minimising the data cleaning time and improving the degree of automation during the data cleaning process, (2) improving the effectiveness of data cleaning. This framework retains the most appealing characteristics of existing data cleaning approaches reviewed in Chapter 2, and improves the efficiency and effectiveness of data cleaning in a database application by introducing two mechanisms: 'algorithm ordering mechanism' (AOM) and 'algorithm selection mechanism'(ASM). In the following sections, the proposed data cleaning framework will be detailed as well as the two mechanisms.

4.2 Data cleaning framework

In this section, the proposed data cleaning framework is introduced, starting with some basic ideas.

4.2.1 Basic ideas

Data cleaning is a labour-intensive, time-consuming, and expensive process, especially when multiple data cleaning activities are involved with huge volumes of data from different data sources during the data cleaning process. Reducing the data cleaning time becomes a motivation in the proposed data cleaning framework.

According to Peng, a data cleaning framework has been proposed which both reduces the cleaning time and maximisesthe degree of automation during the data cleaning process [32]. In Peng's framework, the whole data cleaning process is

firstly broken into two stages to deal with data quality problems associated with a single-source and multi-source respectively. In this way, the time required to deal with multi-source data quality problems is reduced. For example, during the execution of duplicate record detection, comparing each tuple with other tuples will cost lots of time. Reducing the comparison times will save significant time during the process of duplicate record detection. This is achieved by removing the number of tuples to be compared within each single data source.

Further, in each stage, the process is again divided into two sub-processes according to whether or not human involvement is needed to deal with the data quality problem. This helps with minimizing human involvement during the cleaning process and thus, the degree of automation is maximized.

In each sub-process, a strategy is applied to organize the multiple algorithms involved, i.e., algorithms dealing with non-computational-costly errors are put at the front in order [32]. An error of this type requires relative less cleaning time than a computational-costly error. Although this strategy aims to minimize the processing time, the effectiveness associated with these algorithms is not considered. As will be seen later, when effectiveness is considered, a pre-defined order might need to be adjusted with a different algorithm selected. Based on the strategy proposed by Peng, improvements have been made in the proposed framework by introducing two mechanisms (AOM and ASM), where both the effectiveness and efficiency during the data cleaning process are addressed.

Apart from the organization of the multiple algorithms involved during the cleaning process, the selection of a suitable algorithm is also a difficult task [31]. For example, it has been mentioned by Peng that during the data cleaning process, an experienced expert with sufficient business domain knowledge might be required to guide the selection of an algorithm for a particular domain [32]. Since human

involvements during the data cleaning process will minimize the degree of automation, a solution that can automatically select a suitable algorithm according to some rules or strategies is highly expected for a data cleaning approach. In the proposed framework, the selection of an appropriate algorithm is addressed by introducing an ‘algorithm selection mechanism’.

Although it is impossible for us to have a thorough test on all existing available algorithms associated with all dirty data types introduced from the proposed taxonomy, a set of approximate string matching algorithms have been analyzed and evaluated based on different carefully designed databases in chapter 5. With these experimental results, it has shown the possibilities for the proposed ‘algorithm selection mechanism’ to automatically select a suitable algorithm according to the different domain specific pre-defined rules during the data cleaning process. Thus, both effectiveness and degree of automation will be improved.

Finally, the proposed framework has also addressed the DDS problem proposed in chapter 1 by introducing a ‘DDS process’. From the literature, it has indicated that in some cases, cleaning all dirty data types is unrealistic and simply not cost-effective when taking into account the needs of a business enterprise [29]. In the proposed framework, the ‘DDS process’ will help enterprises make a selection of dirty data types by prioritizing the expensive process of data cleaning, therefore maximally benefiting their organizations.

4.2.2 Some definitions

Before the proposed data cleaning framework is detailed, the following definitions are needed regarding the dirty data types and the proposed two mechanisms (AOM and ASM). They will be used during the detailing of the proposed data cleaning framework.

- Single-source error type: An error of this type is present in a single source dataset. For example, missing values, misspelled values, syntax violation, outdated values are all of this type of error. These errors should be cleaned within each single data source before data are integrated from multiple data sources, so that the overall cleaning time will therefore be reduced from cleaning data in multiple data sources. For example, detecting duplicate records from multiple data sources takes time and if a significant number of records can be removed within single sources, the number of comparisons that are necessary for detecting duplicates will be significantly reduced.

- Multi-source error type: An error of multi-source error type is present when data from more than one data source are integrated. For example, the different representation of the values for the same attribute is just a problem of multi-source error type. In one data source, the values ‘F’, ‘M’ are used to represent the attribute ‘Gender’, while in another data source, the values ‘1’ and ‘0’ are used instead. As another example, duplicate records may occur when data are integrated from multiple data sources as the same entity may be represented by an equivalent representation in more than one tuple from different data sources.

- Automatic-removable error type: An error of this type can be detected then corrected without any human involvement. Cleaning this type of error is entirely depending on the selected algorithms. For example, the problem of different representations of ‘Gender’ attribute values in multiple data sources could be detected and corrected with the help of some algorithms automatically without any human interruption.

- Non-automatic-removable error type: A non-automatic-removable error can not be fully detected and then corrected by the algorithms without any

human interruption. For example, during the detection of duplicate records from a single data source or multiple data sources, human involvements sometimes are needed to deal with exceptions such as the missing value in the matching fields. Although some algorithms are available during the detection of duplicate records, an expert sometimes is still required for the final merging task towards the linked records. For example, an expert may need to decide which record should be kept out of the many duplicates and then update the values of the record that is kept and delete the others.

- Computational-costly error type: An error of this type will cost significant time relatively when it is cleaned by an algorithm. Computational-costly is a relative measure. It varies between different types of errors and different algorithms involved. For example, both methods of 'SortingIndex' and 'FullIndex' proposed in Febrl can be applied to detect duplicate records in a dataset. Relatively, 'SortingIndex' algorithm requires less timing cost than the 'FullIndex' algorithm.

- Non-computational-costly error type: An error of this type is an opposite of the computational-costly error. An error of this type can be detected and cleaned without requiring much cleaning time compared with the computational-costly errors.

- Algorithm ordering mechanism (AOM): An algorithm ordering mechanism (AOM) tries to organize the associated dirty data types to be cleaned in a specified order in order to maximize the efficiency and effectiveness during the data cleaning process. This mechanism will be applied to both groups of 'automatic-removable error type' and 'non-automatic-removable error type'. With the help of this mechanism, all dirty data types from each group will be firstly ordered according to the different computational cost associated with

the algorithm selected for each dirty data type. Relatively, non-computational costly errors are put in the front of the order. Later, the entire order will be adjusted according to a further analysis based on the involved algorithms regarding the effectiveness. If an algorithm (A1) has to deal with the values from multiple fields, then any other algorithms existed to improve the quality of the values from these fields are needed to be executed ahead of A1. This mechanism will be further detailed in the case studied later.

- Algorithm selection mechanism (ASM): Algorithm selection mechanism (ASM) helps with selecting a proper algorithm to deal with a specific dirty data type according to different considerations involved during the data cleaning process such as problem domain, error types, error rates, etc. These considerations are presented as the form of pre-defined rules in the system. With the help of ASM, dirty data types will be grouped under two sub-groups namely ‘automatic-removable errors’ and ‘non-automatic-removable errors’ respectively according to the algorithms selected for each dirty data type.

4.2.3 The framework

Briefly, the framework is trying to break the data cleaning process into three stages. Firstly, all dirty data from various data sources are classified into two different groups namely ‘single-source error type’ group and ‘multi-source error type’ group respectively. The first stage and the second stage in the data cleaning process are designed to deal with these two groups of error types exclusively.

In the first stage, dirty data belonging to the group of ‘single-source error type’ is detected and cleaned. Dirty data in this group refers to the dirty data presented in a single data source, e.g., misspelling, and domain constraint violation. In the second stage, dirty data belonging to the group of ‘multi-source error type’ is detected and cleaned. Dirty data in this group refers to the dirty data present when data are

integrated from multiple data sources, e.g., duplicate records. The proposed two mechanisms (AOM and ASM) are then applied to each group during the data cleaning process. In order to improve the degree of automation, dirty data from the two groups are again grouped into two sub-groups namely ‘automatic-removable error types’ and ‘non-automatic-removable error types’ according to whether the dirty data can be fully cleaned without any human involvement.

In the group of ‘automatic-removable error types’, all dirty data should be detected and cleaned by the selected algorithms without any human involvement during the data cleaning process, while in the group of ‘non-automatic-removable error types’, human involvements are required during the cleaning process. In each sub-group, the AOM helps with organizing those associated algorithms selected by the ASM.

In the third stage, the tasks associated with data transformations are performed. Those data cleaned from the first and second stages are ready for the tasks such as instance-level or schema-level format standardizations, data integration, and data aggregation. Finally, data are ready for loading to any database application.

Additionally, before entering into the first stage, a process called ‘DDS process’ can be specified on the dirty data from the various data sources. According to the different needs of an organization, this process helps an organization to select only the most important dirty data to deal with rather than running all algorithms for all possible dirty data candidates in order to minimize the expensive cost associated with the data cleaning process. A general process of the proposed framework is given in Fig.4.1.

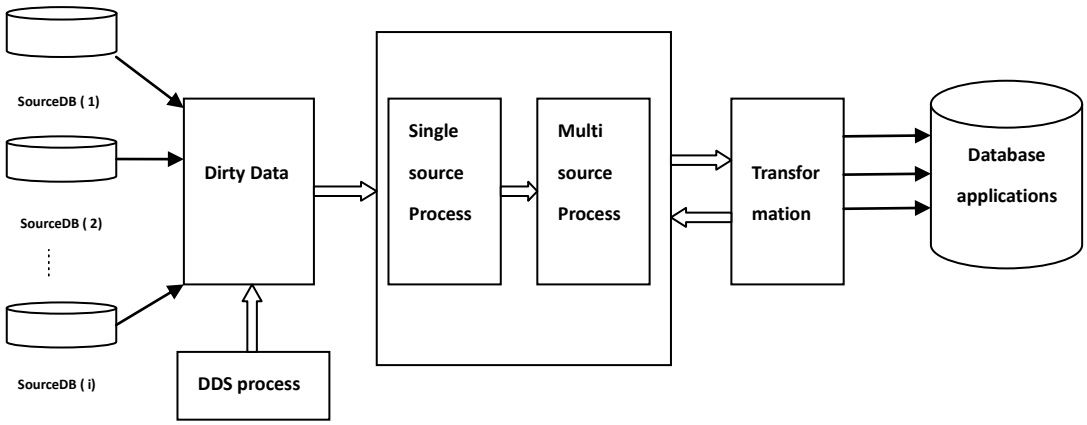


Fig.4.1 A data cleaning framework

All major components and their embedded tasks from the first two stages in Fig.4.1 are detailed below:

(i) The DDS Process

The DDS Process is trying to identify a selection of possible dirty data types rather than focusing on all dirty data types from the different data sources. By only focusing on the selected dirty data types, it is expected that the expensive process of data cleaning can be prioritized and therefore will maximally benefit the organization. Regarding the DDS problem introduced in Chapter 1, when specific needs of a business enterprise have to be taken into account, it is usually not realistic and not cost-effective to clean all the dirty data types encountered from different data sources. Since business rules can be used as guidelines for the validation of information quality, with the help of the proposed rule-based taxonomy, it is reasonable for a business enterprise to pick up a few of the most important groups of business rules rather than all of rules to deal with according to its own business priorities. According to David Loshin, ‘integrating control processes based on data quality rules communicates knowledge about the value of the data in use, and

empowers the business users with the ability to determine how best the data can be used to meet their own business needs'. It also recommended that 'organizing data quality rules within defined data quality dimensions can enable the governance of data quality management and data stewards can use data quality tools for determining minimum thresholds for meeting business expectations, monitoring whether measured levels of quality meet or exceed those business expectations' [12]. The proposed taxonomy of dirty data is a data quality rule based taxonomy which forms relationships between dirty data types and data quality rules. When these data quality rules are organized under the defined data quality dimensions, a relationship between data quality dimensions and dirty data types can also be formed, which will be used to develop a method to deal with data quality problems.

In detail, in order to generate a better DDS process result, an assessment of data quality is first required. According to the review in chapter 2, data quality cannot be assessed independently of the people who use the data, i.e., data consumers. It is possible that the same data used in different tasks may require different quality characteristics. Therefore, both subjective and objective data quality metrics are required during the DDS process.

Firstly, objective assessment is performed. According to the specific business priority policy, data quality dimensions are obtained together with different business rules associated within each dimension. The following five data quality dimensions: accuracy, completeness, consistency, currentness and uniqueness have been used as the dimensions to measure data quality involving data values. Brief introductions of these five dimensions are given below:

- Accuracy dimension: The accuracy of the datum refers to the degree of closeness of its value *v* to some value *v'* in the attribute domain considered correct for the entity *e* and attribute *a*. If the datum's value *v* is the same as a

correct value *v'*, the datum is said to be accurate or correct.

- Completeness dimension: Completeness is the degree to which a data collection has values for all attributes of all entities that are supposed to have values.
- Currentness dimension: A datum is said to be current or up to date at time *t* if it is correct at time *t*. A datum is out of date at time *t* if it is incorrect at *t* but was correct at some moment preceding *t*.
- Consistency dimension: Data is said to be consistent with respect to a set of data model constraints if it satisfies all the constraints in the set.
- Uniqueness dimension: Uniqueness of the entities within a data set implies that no entity exists more than once within the data set.

These data quality dimensions are ordered based on the business priority policy. With the help of the proposed rule-based taxonomy of dirty data, a collection of dirty data types are selected and associated within each data quality dimension. Then, according to different individual needs from different business organizations, the most wanted dimensions are selected and algorithms/methods for dealing with the dirty data types within the selected dimensions are collected.

Meanwhile, a subjective assessment is conducted by different data consumers. Subjective data quality assessment evaluates data quality from views of data collectors, custodians, and data consumers [50] and could adopt a comprehensive set of data quality dimensions which are defined from the perspective of data consumers [4]. The assessment is focussed on the management perspective and concentrates on whether the data is fitness for use. During this process, questionnaires, interviews, and surveys can be developed and used to assess these dimensions. From the literature, subjective assessment results may corroborate with objective assessment results. In this case, the results from objective assessment will be used for the next step. However, when discrepancies exist between the subjective and objective

assessments, organizations should investigate the root causes and consider the corrective actions. For example, whether the dimensions from subjective assessment should be included for the final result or whether the discrepancies can be disregarded for the final result. Finally, with these selected dirty data types along with the available algorithms, a classification is made. These dirty data types together with their algorithms are grouped into the ‘single-source error type’ group and the ‘multi-source error type’ group. The general process of solving the DDS problem is shown in Fig.4.2.

As is shown in Fig.4.2, tasks from the DDS process include:

a) Create an order of the five dimensions according to the business priority policy.

b) Identify data quality problems with the help of the proposed taxonomy of dirty data.

c) Map the dirty data types identified in b) into the dimensions against the classification table.

d) Comparatively analyze on both objective and subjective assessments’ results.

e) Decide dimensions to be selected based on the budget.

f) Select available algorithms, which can be used to detect dirty data types associated with dimensions identified in e).

g) Group the selected dirty data types into ‘single-source error type’ group and ‘multi-source error type’ group with the help of domain and technical knowledge so that they can be dealt with in the next two stages respectively.

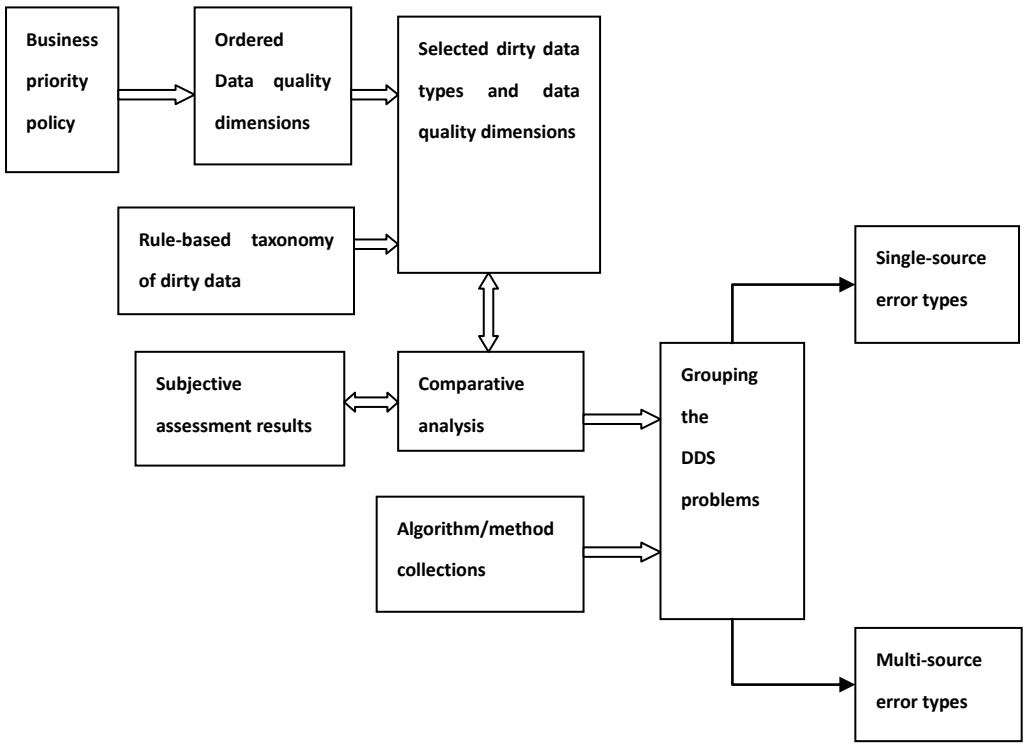


Fig.4.2 The DDS Process

(ii) The single-source process

The purpose of decoupling the process of data cleaning into the two stages of single source process and multi-source process is for the minimization of the data cleaning time. For example, duplicate record is a common dirty data type which occurs when data from multiple sources are integrated together. The most reliable way to detect duplicate records is to compare every record with every other record and is involved with the quadratic cost. It will become even impossible to be accepted when larger sizes of datasets are involved considering the time used for data cleaning. However, if a significant number of tuples can be removed within single sources, the number of comparisons that are necessary for detecting the integrated records will be significantly reduced and the time required will be minimized.

In detail: with all single-source error types identified from source DB(1) to source

DB(i), the ‘algorithm selection mechanism’ is firstly applied to associate an appropriate algorithm to each error type according to the different rules defined by the users.

Then, these single-source error types from each single source DB are again decoupled into two groups namely automatic-removable errors and non-automatic-removable errors. The ‘algorithm selection mechanism’ can help with the decoupling work. For each dirty data type addressed in the proposed framework, the meta data of its related algorithms such as ‘computational cost’, ‘whether a human involvement is needed during the execution’ are kept in the proposed framework. Once a data quality problem is classified in the group of single-source error types and a suitable algorithm is selected by the ‘algorithm selection mechanism’, the corresponding meta data of this algorithm will be extracted and analyzed by the ‘algorithm selection mechanism’.

According to the different meta data supplied for the algorithms involved in the group of ‘single-source error types’, it is easy to decouple these single-source error types into the two sub-groups of ‘automatic-removable errors’ and ‘non-automatic-removable errors’ respectively. For each sub-group, the ‘algorithm ordering mechanism’ is applied to organize the execution of algorithms in each sub-group.

As mentioned, the ordering generated by this mechanism will address both efficiency and effectiveness of data cleaning. Firstly, the meta data of computational cost of each algorithm is extracted and analyzed by the ‘algorithm ordering mechanism’. Algorithms dealing with non-computational cost errors are put at the front in order. Then, the effectiveness associated with each algorithm is further analyzed and re-ordering carried out by this mechanism. Once the ordering work is done, the algorithms are ready to be executed.

The cleaning for the group of ‘automatic-removable errors’ is firstly performed, then followed by the group of ‘non-automatic-removable errors’. As mentioned in section 4.1, algorithm selection and algorithm ordering are two important factors to influence the performance and accuracy of the data cleaning result. Users of existing data cleaning approaches such as Febrl or ARKTOS have to specify the algorithm selection as well as its ordering by themselves, which not only minimize the degree of automation during the data cleaning process but also is likely to result in poor cleaning results as discussed before. The proposed framework provides a solution by introducing two mechanisms (ASM and AOM) to cope with the algorithm selection problem as well as the algorithm ordering problem, which improves the degree of automation during the data cleaning process as well as the efficiency/effectiveness of data cleaning. The general process of dealing with the single-source process is presented in Fig.4.3. As shown in Fig.4.3, in single-source process:

(1) Tasks associated with applying the algorithm selection mechanism include:

a) *For all dirty data types from the group of ‘single-source error types’, selecting an appropriate algorithm for each dirty data type involved.*
b) *Grouping all dirty data types from the group of ‘single-source error types’ based on the selected algorithms into two sub-groups either the sub-group of ‘automatic-removable errors’ or the sub-group of ‘non-automatic-removable errors’.*

(2) Tasks associated with applying the algorithm ordering mechanism include:

a) *Ordering all algorithms from each sub-group of either ‘automatic-removable errors’ sub-group or the ‘non-automatic-removable errors’ sub-group according to the computational cost associated with each selected algorithm. Algorithms dealing with non-computational cost errors are put at the front in the order.*
b) *Adjusting the order obtained from a) to maximize the effectiveness of each*

involved algorithm in the sub-group.

(3) Tasks in the single-source process include:

a) Dealing with dirty data from the sub-group of 'automatic-removable errors' with the help of the selected algorithms by ASM, based on the order generated by AOM.

b) Dealing with dirty data from the sub-group of 'non-automatic-removable errors' with the help of the selected algorithms by ASM, based on the order generated by AOM.

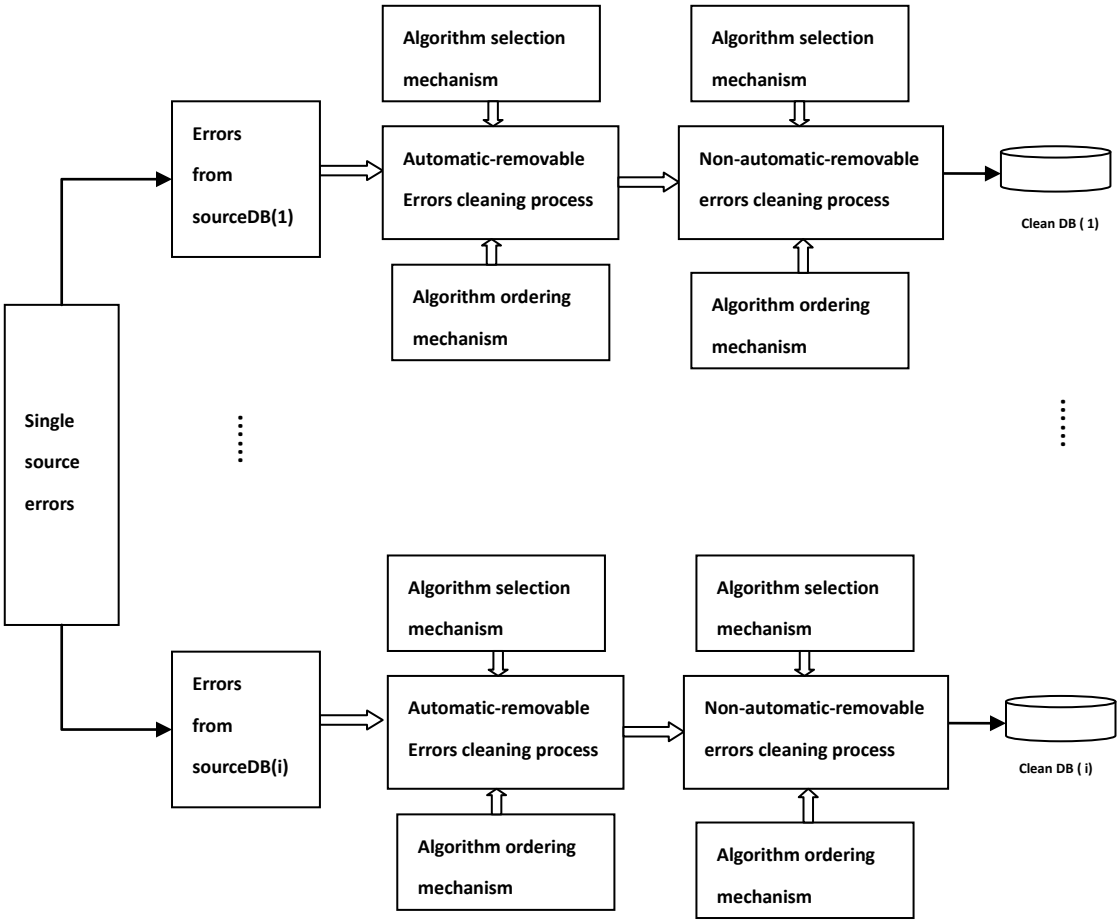


Fig.4.3 The single-source process

(iii) The multi-source process

Multi-source process deals with multi-sources errors. Firstly, with all the multi-source errors identified, ASM is applied to select an appropriate algorithm for each associated dirty data type. Similar to the single-source process, these multi-source errors are then further grouped into two sub-groups: automatic-removable errors and non-automatic-removable errors with the help of ASM.

Dirty data belongs to the sub-group of ‘automatic-removable errors’ are firstly dealt with followed by the dirty data from the sub-group of ‘non-automatic-removable errors’. The AOM is applied to generate an appropriate order to execute the multiple algorithms involved in each sub-group. The general process of dealing with multi-source process is given in Fig.4.4.

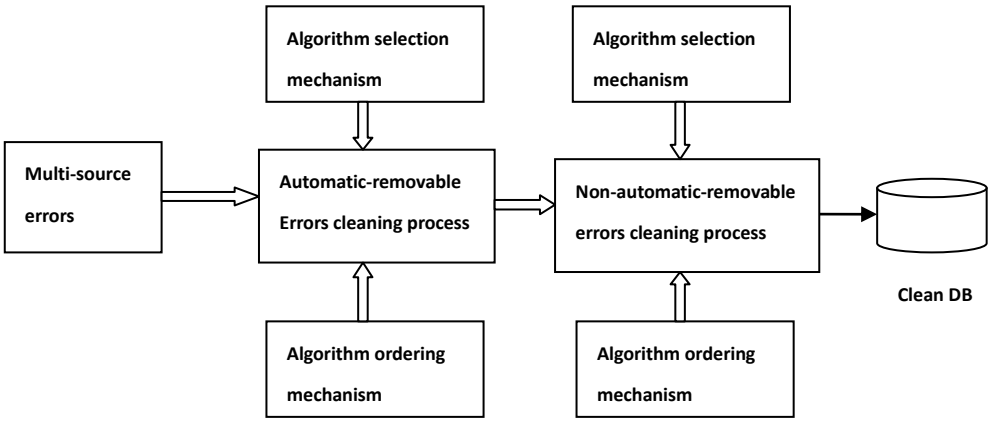


Fig.4.4 The multi-source process

As is shown in Fig.4.4, in the multi-source process:

(1) Tasks associated with applying the algorithm selection mechanism include:

a) For all dirty data types from the group of ‘multi-source error types’, selecting an appropriate algorithm for each dirty data type involved.

b) *Grouping all dirty data types from the group of 'multi-source error types' based on the selected algorithms into two sub-groups either the sub-group of 'automatic-removable errors' or the sub-group of 'non-automatic-removable errors'.*

(2) Tasks associated with applying the algorithm ordering mechanism include:

a) *Ordering all algorithms from each sub-group of either 'automatic-removable errors' sub-group or the 'non-automatic-removable errors' sub-group according to the computational cost associated with each selected algorithm. Algorithms dealing with non-computational cost errors are put at the front in the order.*

b) *Adjusting the order obtained from a) to maximize the effectiveness of each involved algorithm in the sub-group.*

(3) Tasks in the multi-source process include:

a) *Dealing with dirty data from the sub-group of 'automatic-removable errors' with the help of the selected algorithms by ASM, based on the order generated by AOM.*

b) *Dealing with dirty data from the sub-group of 'non-automatic-removable errors' with the help of the selected algorithms by ASM, based on the order generated by AOM.*

Compared with current data cleaning approaches reviewed in Chapter 2, the proposed framework has several features which those existing data cleaning approaches have not considered:

(1) The use of the proposed rule based taxonomy of dirty data during the data cleaning process.

(2) The DDS process, which can help a business take into account the special needs according to the different business priority policy. This will be especially helpful

when the given budget of a business for data cleaning is limited.

(3) The application of an algorithm selection mechanism during the cleaning process. Current data cleaning approaches do not supply any knowledge about the selection of an appropriate algorithm as well as providing a guideline for ordering the selected algorithms. In some approaches, this task is left to its users to make a decision and in others, only a fixed solution for all situations is provided without considering the domain problem or the nature of dirty data. This not only affects the quality of the cleaning result but also increases the cleaning time. From the literature, the selection of a suitable algorithm for dealing with a specific dirty data type has proven to be a difficult task with many aspects need to be considered. Due to the research scope, it is impossible for us to test all algorithms/methods associated with all the dirty data types from the proposed taxonomy. However, in chapter 5, a group of selected approximate string matching algorithms have been used for such a test. As is presented later, a technique that can cope with all situations does not exist.

(4) The application of an 'algorithm ordering mechanism' during the cleaning process. Current data cleaning approaches do not provide guidelines for ordering the multiple data cleaning activities involved during the data cleaning process. Similar to the selection of an algorithm, in some approaches such as ARKTOS, this task is left to its users to make a decision. In others such as Febrl, multiple data cleaning activities have to be done separately rather than in a single data cleaning process. This not only affects the effectiveness of data cleaning results but also minimizes the degree of automation. With the help of the proposed algorithm ordering mechanism, multiple data cleaning activities are organized into a specific order with both degree of automation and effectiveness considered. Thus, the efficiency and effectiveness regarding the whole data cleaning process could be improved.

(5) Decoupling the whole data cleaning process into two sub-processes (single-source process and multi-source process) and dealing with single-source process first followed by the multi-source process. By doing so, the time required for cleaning the dirty data from the group of 'multi-source error types' is greatly reduced and the efficiency of data cleaning should be improved [32].

4.3 A case study

To illustrate the basic framework, a data cleaning tool based on the proposed data cleaning framework has been prototyped. A case study is presented in this section to show the efficiency and effectiveness of the proposed data cleaning framework by applying it to some purposely designed databases.

In the U.K., National Health Service (NHS) is a nationwide organization, which provides health services to all residents. By gathering all information of all residents to a data warehouse (DW), the level of NHS services could be improved. Suppose every city in the U.K. has a single local database which contains the information of residents in the local city. The DW needs to bring altogether the information from each local database in each city. The problem remains that duplicate information may exist either in a single data source or when multiple data sources are integrated. Duplicate information occurs due to many reasons. For example, consider university students who move to another city after graduation. Students from city-A may register their doctors in city-A where their universities are located. These students may again register other doctors in city-B when they move there for their further studies or their new jobs after their graduation. In this case, information on these students may be stored in both cities' local databases which duplicate the information.

The following tables (table 4.1 and table 4.2) show the samples of data entered in the two cities' NHS local databases respectively, where VST stands for valid start time, and VET for valid end time.

No.	Last Name	First Name	Age	City	Post	VST	VET
1	Colae	Liam	22	Edinburgh	Student	22-09-2005	Now
2	Gerrard	John	23	Student	Edinburgh	02-10-2004	Now
3	Higgins	Alan	21	Edinburgh	Student	05-10-2004	20-06-2004
4	Kent	Alex	36	Edinbugh	Engineer	18-09-2003	Now
5	Owen	Mark	18	Edinburgh	Student	06-10-2004	Now
6	Small	Helen	23	Edinburgh	Student	12-09-2002	Now
7	William,Smith		24	Edinburgh	Student	08-10-2004	Now
8	Smith	Mary	34	Edinburgh	Engineer	12-10-2005	10-09-2005
9	Snow	Jamie	22	Edi	Student	10-10-2005	Now
10	Cole	Lieam	22	Edinburgh	Student	22-09-2005	Now

Table 4.1 Records in city-A

No.	Last Name	First Name	Age	City	Post	VST	VET
1	Cole	Liam	26	London	Engineer	20-08-2009	Now
2	Gerrad	John	27	London	Engineer	18-09-2004	Now
3	Higgins	Alan	21	London	Engineer	30-08-2008	Now
4	Kent	John	34	London	Engineer	18-09-2007	Now
5	Owen	Mary	22	London	Student	10-10-2008	Now
6	Small	Helen	23	Lndon	Student	10-09-2003	Now
7	Smith	William	24	London	S	08-10-2008	Now
8	Kirsty	Smith	38	London	Engineer	10-10-2009	Now
9	Snow	John	22	London	Student	08-08-2006	Now

Table 4.2 Records in city-B

By observing the two tables, some dirty data can be easily identified. For example, in some records, the value of 'Edinburgh' has been misspelt as 'Edinbugh' in the 'City' field. NHS has also noticed that some suspicious duplicate personal

information exists in the tables. They decide to detect the duplicate information and eliminate it to make sure the personal information is only kept in the local city's database where the person is currently living. The proposed data cleaning framework has been applied for helping NHS with cleaning such dirty data. The actual cleaning process is detailed below with a DDS process specified firstly before entering into the first stage to deal with dirty data in each single data source.

(1) The DDS process

Data from the two databases has been fully analyzed. Based on the data provided by table 4.1 and table 4.2, the following 9 dirty data types are identified according to table 3.8:

- ***DT.15****: in table 4.1, a missing value is observed in the field of 'First Name' where 'No.'=7.*
- ***DT.17****: in table 4.1, in the record where 'No.'=7, the value of its field 'First Name' has entered into the field of 'Last name' followed by the correct last name value, which causes a problem of 'Extraneous data entry'.*
- ***DT.20****: in table 4.1, in the record where 'No.'=2, the problem of 'entry into wrong field' is noticed. The value of its field 'City' has entered into the field 'Post' and vice versa. Similarly, in table 4.2, in the record where 'No.'=7, the value of field 'First Name' has entered into its field 'Last Name' and vice versa.*
- ***DT.23****: in table 4.2, suppose it is known that 'Alan Higgins' in table 4.2 is exactly the same person in table 4.1, then an outdated value is observed in the field of 'age' in the record where 'No.'=3.*
- ***DT.27****: in table 4.1, misspelling errors are seen in records where 'No.' =1, 4, and 10 in the fields of 'Last Name', 'City', and 'First Name' respectively. In table 4.2, misspelling errors are seen in records where 'No.' =6 and 8 in the*

fields of 'City' and 'Last Name' respectively.

- ***DT.32**: in table 4.1, in the record where 'No.'=9, the value in the field 'City' is abbreviated as 'Edi' rather than 'Edinburgh'.*
- ***DT.33**: in table 4.2, in the record where 'No.'=7, the value in the field 'Post' is represented with a special character 'S' rather than 'Student'.*
- ***DT.28**: in table 4.1, suspicious duplicates are observed. For example, records where 'No.'=1 and 10.*
- ***DT.29**: when data are viewed from both tables, many suspicious duplicate records are noticed. For example, records where 'No.'=1, 2, 3, and 7 from table 4.1 are all suspicious candidates.*

In order to build a high quality DW, the data in the databases should be cleaned as much as possible. Thus, NHS plans to execute the following data cleaning activities (DCA) to improve its data quality:

No.	Data Cleaning Activity
DCA.1	Detect/fill missing values from the fields of 'Last Name' and 'First Name'
DCA.2	Standardize values from the fields of 'Last Name' and 'First Name'.
DCA.3	Correct the values from entering into wrong fields based on the fields of 'City', 'Post', 'First Name', and 'Last Name'.
DCA.4	Update values from the field of 'Age'.
DCA.5	Correct the misspelt values in 'Last Name', 'First Name' and 'City' fields.
DCA.6	Detect/standardize the abbreviated values from the field of 'City'.
DCA.7	Detect the use of special character values from the field of 'Post' and correct them to the standardized values.
DCA.8	Clean the duplicate records from each single dataset.
DCA.9	Clean the duplicate records from the integrated datasets.

Table 4.3 Data cleaning activities

Suppose due to the limited available resources within the NHS, performing all the 9 data cleaning activities is unrealistic. Therefore, before entering into the first stage to deal with the dirty data from different single data sources, the DDS process is performed to help NHS with selecting some of most important dirty data to clean. The DDS process for NHS begins with a mapping between business rules from NHS and data quality dimensions.

(i) Mapping between Data Quality Rules and Data Quality Dimensions

With the five data quality dimensions introduced in section 4.2.2, a new classification of the dirty data types is introduced beginning with a mapping of data quality rules with data quality dimensions. Table 4.4 shows the result of the mapping:

Data Quality Dimension	**Data Quality Rules**
Accuracy dimension	R2.2, R3.2, R4.2, R4.4
Completeness dimension	R1.3, R4.1
Currentness dimension	R4.3
Consistency dimension	R1.2, R3.1, R4.6
Uniqueness dimension	R4.5

Table 4.4 Data quality dimension and data quality rules

(ii) A Classification

The result of Table 4.4 provides an immediate help for the proposed classification of dirty data within the new taxonomy. Combining the result from table 3.9 and table 4.4, a classification of dirty data types based on data quality dimensions is achieved as shown in table 4.5.

Data Quality Dimension	Dirty Data Type
Accuracy dimension	DT.5~DT.9, DT.11~DT.13, DT.16~DT.20, DT.25~DT.27
Completeness dimension	DT.3, DT.4, DT.14, DT.15
Currentness dimension	DT.21~DT.24
Consistency dimension	DT.1, DT.2, DT.10, DT.30~DT.38
Uniqueness dimension	DT.28, DT.29

Table 4.5 Data quality dimensions and dirty data types

Therefore, data cleaning activities for NHS can be considered as cleaning dirty data by different data quality dimensions. The DDS problem described in Chapter 1 can therefore be solved by forming a relationship between the defined data quality dimensions and dirty data types with the help of the rule-based taxonomy of dirty data.

(iii) The Method and the Result

With the help of the classification from part (ii), the method described in section 4.2.2 is prototyped during the DDS process below:

a) Create an order of the five dimensions according to the business priority policy.

According to NHS's priority policy, the uniqueness of information recorded in DW is often very stringent for NHS data following with the accuracy. The order of the five data quality dimensions for NHS is therefore: Uniqueness → Accuracy → Consistency → Completeness → Currentness, descending in priority.

b) Identify data quality problems.

With the help of the rule-based taxonomy of dirty data, the following 9 dirty data types are identified: DT.17, DT.20, DT.27, DT.32, DT.33, DT.15, DT.23, DT.28, DT.29.

c) Map the data types identified in b) onto the dimensions against the classification table.

With the help of table 4.5, the mapping is achieved as given in table 4.6. It is clear that all identified data quality problems have been organized under all the five data quality dimensions. Suppose it is impossible for NHS to address all data quality problems associated with the five data quality dimensions. Therefore, the problem that NHS is facing is how to select a group of dirty data types to deal with, the DDS problem.

Data Quality Dimension	**Dirty Data Type**
Uniqueness dimension	DT.28, DT.29
Accuracy dimension	DT.17, DT.20, DT.27,
Consistency dimension	DT.32, DT.33,
Completeness dimension	DT.15
Currentness dimension	DT.23

Table 4.6 An example of data quality dimensions and dirty data types

d) Decide dimensions to be selected based on the budget.

According to the priority policy, NHS chooses to deal with the data quality problems related with uniqueness dimension and accuracy dimension firstly since the uniqueness dimension and the accuracy dimension are much more urgent than other

dimensions. Thus, dirty data types: DT.17, DT.20, DT.27, DT.28, and DT.29 are chosen firstly to be cleaned.

e) Select the available algorithms, which can be used to detect dirty data types associated with dimensions identified in d).

f) Group the selected dirty data types into single-source errors group and multi-source errors group with the help of domain and technical knowledge so that they can be dealt with in the next two stages respectively.

Finally, the DDS process provides the following data cleaning activities, the related dirty data types and the available techniques for NHS to choose (see table 4.7).

Group	**DCA**	**DDT**	**Techniques**
Single-source error types	DCA.3	DT.20	Look-up tables; reference functions
	DCA.5	DT.27	Spell checker
	DCA.2	DT.17	Pattern learning technique
	DCA.8	DT.28	Deduplication techniques; Approximate string matching algorithms
Multi-source error types	DCA.9	DT.29	Recode Linkage techniques; Approximate string matching algorithms

Table 4.7 The grouping results

(2) Dealing with the single-source process

According to table 4.7, four data cleaning activities are involved in the single-source process and four dirty data types: DT.20, DT.27, DT.17 and DT.28 belong to the group of ‘single-source error types’. For each dirty data type, available algorithms are provided. The proposed data cleaning approach will firstly deal with these data

cleaning activities. For each dirty data type involved in this group, the ‘algorithm selection mechanism’ is applied to associate an appropriate algorithm out of the other alternatives. Then, this mechanism further decouples these dirty data types into two sub-groups according to the metadata provided by each algorithm, i.e., ‘automatic-removable errors’ and ‘non-automatic-removable errors’ (see table 4.8). For example, in this case study, a duplicate detection method ‘FullIndex’ has been associated with dirty data type DT.28 to detect duplicate records in each single data source. However, since duplicate detection based on this method some times requires an expert’s involvement to determine the matching result, DT.28 is grouped into the ‘non-automatic-removable errors’.

Group	**Sub-group**	**DDT**	**Techniques**
Single-source error types	Auto	DT.20	Reference function
		DT.27	Spell checker
	Non-auto	DT.17	standardization
		DT.28	FullIndex, String Matching Algorithm

Table 4.8 An example of the two sub-groups

With the two sub-groups, the ‘algorithm ordering mechanism” is applied to automatically specify a sequence of the execution order regarding the multiple algorithms involved. This function tries to manage the ordering of the multiple algorithms based on two considerations.

The first consideration is related to the efficiency of data cleaning. From this point of view, an order is arranged aiming to reduce the processing time. According to the ‘algorithm ordering mechanism’, dirty data types from the group of ‘automatic-removable errors’ are firstly cleaned with the supplied algorithms. In this example, DT.20 and DT.27 belongs to the group of ‘automatic-removable errors’

since the techniques provided by the system can automatically detect these errors and correct them to the expected values. Dirty data types from the group of 'non-automatic-removable errors' are dealt with secondly. Dealing with these dirty data sometimes requires involvement from an expert. For example, during a duplicate elimination process, an expert has to be involved to make a decision for the merging or purging tasks. In this example, DT.17 and DT.28 belongs the group of 'non-automatic-removable errors'. In each sub-group, the 'algorithm ordering mechanism' is further applied to organize the selected algorithms. Algorithms dealing with non-computational cost errors are put at the front in the order according to the strategy proposed in the 'algorithm ordering mechanism'. This strategy aims to minimise the processing time and also ensures that when it's time to perform computational costly errors, such as duplicates, the data volume should be significantly reduced. Hence, the processing time needed is reduced [32]. With this first consideration, the order of the execution of data cleaning regarding the involved dirty data types in the group of 'single-source error types' will be: DT.20→DT.27→DT.17→DT.28.

The second consideration for the 'algorithm ordering mechanism' is related to the effectiveness of the data cleaning. From this point of view, the nature of the selected algorithm as well as the associated data cleaning rules are involved to arrange an order. In this example, four data cleaning activities are associated within the single-source process. For each data cleaning activity of DCA.2, DCA.3, and DCA.5, executing the associated algorithm will only affect the field values where the corresponding dirty data reside. With respect to the effectiveness of data cleaning for the three dirty data types, the order based on the first consideration will not cause any side-effect. However, suppose the Jaro algorithm for DCA.8 is applied on the fields of 'Last Name' and 'First Name' to calculate the similarities of the name values and the rule for duplicate detection is defined as "if the names of the two records are similar and the value of the field 'Post' is 'Student', then the two records

are duplicate records". Clearly, according to this rule as well as the application of the Jaro algorithm, data values of the three fields 'Last Name', 'First Name' and 'Post' are all involved.

Obviously, poor data quality of the three fields will result in a low effectiveness of the duplicate detection in this case. From table 4.3 and table 4.5, it is known that the existence of dirty data types: DT.15, DT.17, DT.20, DT.27 and DT.33 will influence the data quality in the three fields. Therefore, in order to improve the effectiveness regarding the cleaning of DT.28, the two extra dirty data types DT.15 and DT.33 should be considered before DT.28.

In this case, the 'algorithm selection mechanism' is again applied to find a suitable algorithm for the two dirty data types and grouped them into the two sub-groups again. The order generated by the 'algorithm ordering mechanism' for the single-source process is: DT.15→DT.33→DT.20→DT.27→DT.17→DT.28 based on the two considerations.

However, suppose due to the large dataset involved in the single-source process, the 'SortingIndex' method is selected by the 'algorithm selection mechanism' to deal with DT.28 rather than the 'FullIndex' method. By applying the 'SortingIndex' algorithm, the cost is reduced by compromising the accuracy. During the execution of the 'SortingIndex' algorithm, the creation of the sorting key is an important step. Table 4.9 shows an example of how the sorting key is created.

First Name	Last Name	Address	ID	SortingKey
John	Smith	456 Merchiston Crescent	45678987	SmiJoh456Mer456
John	Smyth	456 Merchiston Crescent	45688987	SmyJoh456Mer456
Smith,	John	456 Merchiston Crescent	45688987	JohSmi456Mer456

Table 4.9 An example of sorting key

The 'SortingIndex' algorithm is proposed by Hernandez *et al* which sorts the records based on the sorting key values [13]. The sorting key value for each record is computed by extracting relevant attributes or portions of the attributes' values. Relevant attributes should have sufficient discriminating power in identifying records that are likely to be duplicates. Table 4.9 shows an example of how sorting key values might look. The sorting key values are a combination of sub-string values of attributes in the order of last name→first name→address→ID. Since the records are sorted using the sorting key values generated, the attribute that first appears in the sorting key selection has the most discriminating power, followed by the attributes that appear subsequently with decreasing discriminating power. Therefore, it is important that the value of the first selected attribute should be as clean as possible.

In table 4.9, all three records are supposed to be the duplicate records. However, because the first name value and the last name value of the third person have been transposed (DT.34), the third record might not end up in the same sliding window as the first and second records. Records not in the same window will not be considered as candidate duplicate record pairs and the final cleaning result will be defective. Therefore, in the case that a 'SortingIndex' method is applied to detect duplicate records, it is necessary to make sure the attributes' values that are used for creating the soring key values should be as clean as possible.

In the case proposed by table 4.9, suppose a method of dealing with inconsistency problem (DT.34) for personal names is firstly applied before applying the 'SortingIndex' algorithm, the sorting key value of the third record will be corrected as 'SmiJoh456Mer456' which is exactly the same as the first record's sorting key value. All three records in table 4.9 in this case will fall into the same window and will compare to each other. With the selected name matching algorithm applied later, the tree records might be detected as the duplicate records as expected.

In the NHS case, suppose in order to deal with DT.28, the sorting key values for the 'SortingIndex' method are a combination of sub-string values of attributes in the order of City→Last Name. The Jaro algorithm is still selected by the 'algorithm selection mechanism' to calculate the similarities of the values from the fields of 'Last Name' and 'First Name'. The cleaning rule for DT.28 is changed to "if the names of the two records are similar and the value of the field 'City' is the same, then the two records are duplicate records".

In this case, the data quality of the values from the field 'City' is much more important than the field of 'Post'. Therefore, the order this time is changed to 'DT.15→DT.32→DT.20→DT.27→DT.17→DT.28'

(3) Dealing with the multi-source process

In the group of 'multi-source error types', the cleaning process is similar to the group of 'single-source error types'. When multiple dirty data types are involved, the 'algorithm ordering mechanism' is firstly applied to find a suitable algorithm for each dirty data type and then further groups these dirty data types into the two sub-groups of 'automatic-removable errors' and 'non-automatic-removable errors'. In each sub-group, the 'algorithm ordering mechanism' is used for guiding the sequence of the multiple algorithms involved. In this example, only DT.29 is

involved in the multi-source process. According to the ‘algorithm selection mechanism’, the method of ‘FullIndex’ is selected to detect the duplicate records from the multiple data sources with the help of the Jaro algorithm to calculate the similarities of both name values. For any pair of detected duplicate records, a confirmation rule is defined as ‘for each pair of records detected, if the value of the field ‘Post’ is *Student* in one record, and the difference of the values in the field of ‘VST’ is 4 or more years, then the two records are confirmed as duplicates’. In this case, regarding the effectiveness of the involved algorithm, the following order is made against on the following dirty data types: DT.15→DT.33→DT.20→DT.17→DT.29. When all cleaning activities are finished, the data of NHS are ready for the final stage of ‘transformation process’ where data are transformed into the expected formats according to the different requirements from the DW.

The case study in this section clearly shows the complexities regarding the organization of the multiple data cleaning activities during a data cleaning process. It shows that many factors are needed to be considered for specifying an order towards the multiple cleaning activities involved such as the nature of the selected algorithm, the cleaning rules, the computational cost.

A good order of data cleaning activities not only will improve the efficiency of the data cleaning but also will improve the effectiveness. Unfortunately, according to the author’s knowledge, none of the existing data cleaning approaches reviewed in chapter 2 has addressed this problem intentionally. Only one approach (ARKTOS) has mentioned organizing of the multiple cleaning activities before the cleaning process. However, organizing the multiple cleaning activities in ARKTOS totally depends on the user’s individual preference without specifying the many factors as the proposed framework does. For users who are not familiar with the different problem domains as well as the cleaning techniques involved, a poor order of the

multiple cleaning activities might be made. This will result in a low efficiency and low effectiveness data cleaning result during the data cleaning process.

Existing approaches such as AJAX and IntelliClean provide only a fixed cleaning process from ‘data standardization’ to ‘duplicate elimination’ without giving any further details in each step. Additionally, IntelliClean offers only one fixed solution to deal with a cleaning activity. For example, only SNM algorithm is provided in IntelliClean to detect duplicate records. The ‘algorithm selection mechanism’ from the proposed framework is similar to the mechanism used in the physical level in AJAX. However, compared with AJAX, the proposed framework addresses more concerns regarding the selection of a suitable algorithm. In AJAX, in order to compare two records, users are required for some specifications such as specifying the required algorithms manually such as specifying an approximate string matching algorithm and setting the related parameters manually.

However, as is shown later in chapter 5, users who are not familiar with the selection of a suitable algorithm will usually make a wrong choice among the many alternatives, which will result in a low timing performance and low accuracy performance with respect to the final cleaning results. In the proposed framework, both timing performance and accuracy performance are considered by addressing the two proposed mechanisms during the data cleaning process.

4.4 Conclusion

In this chapter, a novel data cleaning framework has been proposed, which aims to challenge the following issues: (i) minimising the data cleaning time and improving the degree of automation in data cleaning, (ii) improving the effectiveness of data cleaning. Additionally, the proposed framework provides a function (The DDS process) to address the special case when individual business requirements are

involved. This function can help a business to take into account special needs according to different businesses priority policies.

The proposed framework retains the most appealing characteristics of existing data cleaning approaches, and improves the efficiency and effectiveness during the data cleaning process. Compared with existing data cleaning approaches mentioned in chapter 2, the proposed framework provides several exclusive features which have not been addressed in those approaches.

Firstly, regarding the ability for a data cleaning approach to deal with the various dirty data types, the proposed framework tries to address as many dirty data types as possible according to the proposed taxonomy of dirty data. Existing approaches only focus on solving two kinds of data cleaning activities, i.e., data standardization and duplicate records elimination. Some tools such as ARKTOS only focus on solving one activity. Obviously, none of the existing tools in section 2.2 can help with providing an all-in-one solution to the problem proposed in the case study in section 4.3.

Secondly, the proposed framework addresses the order of the various cleaning activities exclusively and provides an automatic solution to organize the sequence of these activities, i.e., 'algorithm ordering mechanism'. None of the existing data cleaning approaches from chapter 2 has addressed this problem. In section 4.1, an order regarding six data cleaning activities is proposed by Müller and Freytag. However, in the proposed case study, both orders generated by the 'algorithm ordering mechanism' are different from the one given by by Müller and Freytag. For example, according to the order given by Müller and Freytag, dealing with missing values is after the format adaptation. In the proposed case study, dealing with missing values is performed before the format adaptation due to the consideration of computational cost and effectiveness associated with the involved algorithms. To

some extent, the order given by Müller and Freytag is a general order to maximize the effectiveness of the data cleaning without considering the efficiency during the data cleaning process. The order proposed by the 'algorithm ordering mechanism' addresses both effectiveness and efficiency during the data cleaning process.

Finally, the proposed framework supplies a function of 'algorithm selection mechanism' which provides an optimized algorithm regarding the different factors involved such as problem domain, error rate and computational cost. It selects an optimized algorithm to deal with different problems with various factors involved. In this way, both effectiveness and degree of automation are improved.

In the next chapter, experiments are designed regarding the selection of a suitable approximate string matching algorithm. With the achieved experimental results, the importance regarding the selection of a suitable algorithm during the data cleaning process is highlighted.

CHAPTER 5 EXPERIMENT AND EVALUATION

In chapter 4, the proposed data cleaning framework was presented. The special feature exclusively designed for the proposed framework is the introduction of two mechanisms during the data cleaning process. Regarding the selection of a suitable algorithm for each data cleaning activity, currently a user is required to manually select an algorithm for a specific cleaning task in existing data cleaning approaches. For example, Fig.5.1 shows a list of available approximate string matching algorithms provided by Febrl for the user to choose during the process of duplicate record detection.

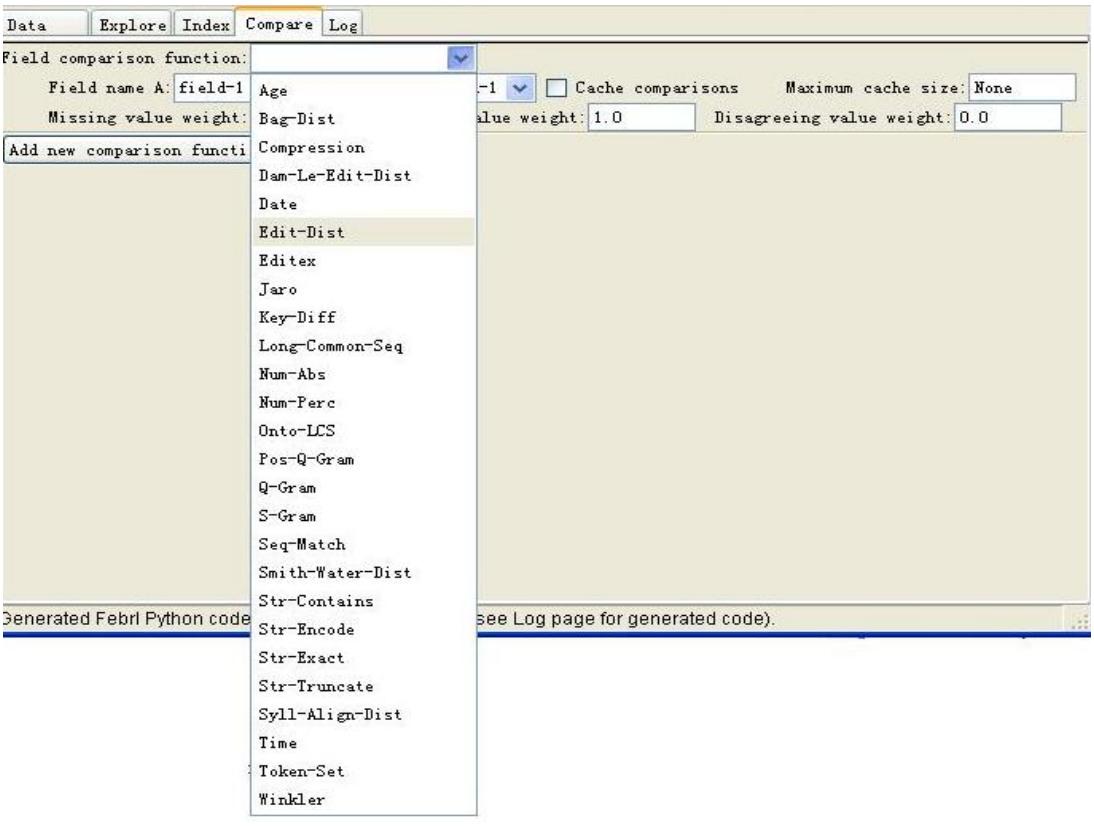


Fig.5.1 Approximate string matching algorithms from Febrl

Compared with other existing tools which only adopt one fixed method to deal with a cleaning task without considering the different problem domains, Febrl is an

advance by providing multiple solutions for a cleaning task to each of the problem domains. Still, according to the review work in chapter 2, the disadvantage is clear that for users who do not have enough knowledge about these techniques, making a selection out of multiple alternatives is a difficult task. Even when an algorithm such as an approximate string matching algorithm is selected for a special task, how to set its related parameters is still unclear in existing approaches. For example, approximate string matching is an important step during the duplicate record elimination process and setting a threshold value for the selected approximate string matching algorithm is an important part for the matching task.

Traditionally, in existing approaches such as AJAX, a universal threshold value is used for the selected approximate string matching algorithm without concern for the different problem domains, characteristics associated with the dataset such as error rate or size of dataset. As discussed, threshold value is one of the many factors to affect the effectiveness of the matching result and many factors should be considered when setting its value. In order to improve the degree of automation as well as the effectiveness of data cleaning, the ‘algorithm selection mechanism’ is provided in the proposed data cleaning framework to facilitate the user in selecting a suitable algorithm for a specific cleaning task.

Although it is impossible for us to have a thorough test on all existing available algorithms associated with the various dirty data types introduced in the proposed taxonomy, a set of approximate string matching algorithms have been analyzed and evaluated based on different carefully designed databases. Experimental results confirm the statement that there is no clear best technique for all situations. Suggestions have been made, which can be used as guidelines for researchers and practitioners to select an appropriate matching technique for a given dataset. Thus, both effectiveness and degree of automation will be improved during the data cleaning process.

5.1 Introduction

String data is by nature more prone to contain errors such as misspelling, different representations due to abbreviations, different word sequence, and use of alias name values. Therefore, expressions denoting a single entity may be different from word sequence, spelling, spacing, punctuation and use of abbreviations. For example, the same person's name can be referred as 'John Smith', 'Smith, John', 'J.Smith'. Records that describe the same entity might differ syntactically due to containing such expressions. They can be found in a single dataset or when they are integrated from multiple data sources, which are termed as 'Duplicate record in single data source' (DT.28) or 'Duplicate record in multi data source' (DT.29) according to the proposed taxonomy of dirty data.

The problem of identifying such duplicate records in databases is an essential and challenging step for data cleaning and data integration. This problem has become a crucial problem as more and more data stored in database systems needs to be integrated for the purpose of supplying decision support with the help of database applications. In data cleaning, the task of dealing with duplicate records is addressed by record matching techniques, also known as merge-purge, data de-duplication and instance identification. Generally, record matching can be defined as the process of identifying records in the same or different databases that refer to the same real-world entity. From the literature, there are two types of record matching: structural heterogeneity related and lexical heterogeneity related.

Structural heterogeneity belongs to the schema-level problem, which refers to the problem of matching two databases with different domain structures. For example, the value of a customer's home address might be stored in a single attribute 'address' in one database but might be presented in another database with more attributes such as 'street', 'city', and 'postcode' respectively. Lexical heterogeneity belongs to the

instance-level problem which refers to databases with similar structure but different representation of data. For example, consider a personal name value. A name value of a person may be represented as “John Smith” in a record from one data source and as “John Smtih” in a record from another data source containing a misspelling. When data are integrated from different data sources, the two records are treated as two different persons and a duplicate record problem occurs. In this research, the lexical heterogeneity problem is foremost and it is assumed that the schema-level structural heterogeneity has been resolved as a priori. As shown in this lexical heterogeneity example, if these two names are not treated as the same person then it might introduce a duplicate error when integrating the data from these two sources.

Names are important pieces of information when databases are de-duplicated. From the literature, name matching can be defined as the process of determining whether two name strings are instances of the same name. As mentioned in the beginning of this section, many reasons can cause name variations such as typos during data entry, use of different name formats, use of abbreviations. Thus, exact name comparison is not able to generate a good matching result. Rather, an approximate measure of how similar two names are is expected.

Name matching in databases has been a persistent and well-known problem for years [118]. From the literature, several techniques are available to deal with this problem [14, 119-123]. However, still, there is no clear best technique for all situations [124]. A problem still exists for researchers and practitioners as how to select a technique for a given dataset [125]. In the past decade, several researchers have challenged this problem [124, 126-128]. However, none of them have undertaken a comprehensive analysis and comparison that considers the effect on the performance of accuracy and timing caused by the following factors: error rates, type of strings, type of typos, and the size of datasets. An overview of this work is given below in section 5.2.

5.2 Related work

Bilenko *et al* evaluated and compared a few effective and widely used approximate string matching algorithms for matching strings' similarity [126]. Broadly, these algorithms can be classified into two categories namely ‘character-level algorithms’ and ‘token-level algorithms’.

Character-level algorithms are designed to handle typographical errors. However, it is often the case that typographical conventions lead to the rearrangement of words, e.g., “John Smith” vs. “Smith, John”. In such cases, character-level algorithms fail to capture the similarity of the entities. Token-level algorithms are designed to compensate for this problem. Therefore, character-level algorithms are good for the single word problem, while token-level algorithms for the matching with more than one word.

In Bilenko *et al*’s work, five character-level algorithms and three token-level algorithms are used for the experimental works. 11 different datasets were used and the sizes of the 11 datasets ranged from 38 records to 5709 records. For each dataset, a single string formed with different sub-strings concatenated from multiple fields is used to evaluate the matching effectiveness of the selected algorithm. Based on the experimental results, the authors claim that Monge-Elkan algorithm performs best on average and SoftTF-IDF performs best overall. However, as pointed out by the authors, individual algorithm’s performance varies significantly when different datasets are considered. For example, in the ‘Census dataset’, the simple ‘Levenshtein’ algorithm performs the best while it is the worst on average. Even methods that have been tuned and tested on many previous matching problems can perform poorly on new and different matching problems.

A further examination of the problem reveals that an estimate of similarity between

strings can vary significantly depending on the domain for each field under consideration. During the experiment, each record is treated as a single, long-string field and the measurement of the similarity is just calculated based on the alignment strings from the different fields. Although the authors have also confirmed the reason associated with the different domains involved in the different fields, further examinations are not achieved in this work. The authors confirm that the limitation of the selected algorithm is the absence of special knowledge of the specific problem at hand and the solution is that some knowledge of the problem should be introduced to the algorithm used. However, this confirmation is not further expanded in detail in their work. Besides, regarding the threshold value used for the evaluation of an algorithm, only a suitable threshold value is chosen for the test as mentioned in the work. The work does not mention what a suitable threshold value should be for each algorithm and whether or not the value is universal for all the eight selected algorithms.

Christen [124] tested more algorithms and provided a comprehensive analysis and comparison among these algorithms specifically to personal names. Christen discussed the characteristics of personal names as well as the potential sources of variations of personal names in detail. A number of algorithms that can be used to match personal names are reviewed. The author evaluated both accuracy and timing performance of the selected algorithms, considering given names, surnames and full names respectively. Based on the experimental results, the author claims that no single algorithm performs better than all others and nine useful recommendations regarding the algorithm selection during the matching of personal names are proposed in this work. Table 5.1 presents these 9 recommendations from Christen's work.

No.	Recommendations
1	It is important to know the type of names to be matched, and if these names have been properly parsed and standardized, or if the name data potentially contains several words with various separators.
2	If it is known that the name data at hand contains a large proportion of nicknames and similar name variations, a dictionary based name standardization should be applied before performing the matching.
3	Phonetic encoding followed by exact comparison of the phonetic codes should not be used. Pattern matching techniques result in much better matching quality.
4	For names parsed into separate fields, the Jaro and Winkler techniques seem to perform well for both given and surnames, as do uni- and bigrams.
5	The longest common sub-string technique is suitable for unparsed names that might contain swapped words.
6	Calculating a similarity measure with respect to the length of the shorter string (Overlap coefficient) seems to achieve better matching results (compared to using the Dice coefficient or Jaccard similarity).
7	The Winkler modification (increase similarity when name beginnings are the same) can be used with all techniques to improve matching quality.
8	A major issue is the selection of a threshold that results in optimal matching quality. Even small changes of the threshold can result in dramatic drops in matching quality. Without labelled training data, it is hard to find an optimal threshold value. Optimal threshold values will also vary between data sets.
9	If speed is important, it is imperative to use techniques with time complexity linear in the string length (like q-grams, Jaro, or Winkler), as otherwise name pairs made of long strings (especially unparsed full names) will slow down matching. Alternatively, filtering using bag distance followed by a more complex edit distance based approach can be used.

Table 5.1 Recommendations by Peter Christen

According to the author, regarding name matching, there is no single best algorithm available. Particularly, the author pointed out the importance of choosing a suitable threshold value (see recommendation 8). It is argued that the selection of a proper threshold value is a difficult task, even small changes of the threshold value could result in dramatic drops in matching quality. Although it is believed by the author that characteristics of both name data to be matched as well as the algorithms are needed to be considered when selecting an algorithm, more detailed analysis into these characteristics are not further studied but left as future work.

Hassanzadeh *et al* [128] presented an overview of eight approximate string matching algorithms and evaluated their effectiveness on several carefully designed datasets. Regarding the algorithms studied in this work, only one algorithm belongs to the character-level algorithm, i.e., Levenshtein algorithm. The other seven algorithms all belong to the token-level algorithms. As is pointed by the author, the effectiveness of the algorithm highly depends on the characteristics of the data, which is also mentioned by Christen.

Hassanzadeh *et al* designed a set of datasets for the experimental work and data from each dataset are associated with an exclusive specific characteristic such as the error rate (the amount of errors) or the type of errors. The problem regarding the threshold value selection proposed by Christen is studied in this work. According to the experimental results, the authors claim that the higher the error rate, the lower the threshold value should be set. Regarding the evaluation of the relative effectiveness of the algorithms, characteristics such as data error rate and data error type are involved in the evaluation work. According to the authors, both data error rate and error type will influence the effectiveness of the selected algorithm. For example, according to the experimental result, when the error rate of a given dataset is high, the HMM algorithm is among the most effective algorithm and could be selected to perform the matching task. However, when the error rate is low, the

effectiveness of HMM algorithm becomes low and should not be selected for the matching task.

Compared with the previous work, Hassanzadeh *et al* mainly focus on the token-level algorithms. Besides, regarding the characteristics of data involved in the experimental work, only two characteristics are considered, i.e., data error rate and error type. The sizes of all the datasets used for the experiment are all the same. Compared to other work, at least two questions are still unclear: if the conclusion regarding the effectiveness of algorithms proposed by the authors based on the experimental results can also apply to the character-level algorithms and if different sizes of datasets will influence the effectiveness of the selected algorithms.

In this chapter, five approximate string matching algorithms are reviewed in section 5.3 and their performances are evaluated against the following characteristics: the error rate in a dataset, the different threshold value chosen, the selected type of strings in a dataset, the type of typo in the selected strings and the size of a dataset. The experiments are performed based on a set of carefully designed datasets. Compared to Hassanzadeh *et al*, the five algorithms used in this research are all character-level algorithms in order to answer whether the conclusion made by Hassanzadeh *et al* will also apply to the character-level algorithms. The experimental results provide an opportunity to help with selecting a suitable algorithm for a name matching task, which can improve the efficiency and effectiveness during the process of duplicate record detection.

5.3 Matching techniques

Name matching can be defined as “the process of determining whether two name strings are instances of the same name” [129]. In this research, the proposed experiments are focused on single name values and five popular character-level

algorithms namely ‘Levenstein’, ‘Smith-Waterman’, ‘Jaro’, ‘Jaro-Winkler’ and ‘Q-Gram’ are selected for the experiments.

(i) *Levenshtein*

The Levenshtein distance [121] is defined to be the minimum number of edit operations required to transform string s_1 into s_2. Edit operations are:

- Delete a character from string s_1.
- Insert a character in string s_2 that does not appear in s_1.
- Substitute one character in s_2 for another character in s_1.
- Copy one character from s_1 to s_2.

The distance (number of edits) between two strings s_1 and s_2 can be calculated based on an efficient scheme for computing the lowest-cost edit sequence for these operations. This can be realized using dynamic programming techniques. The Levenshtein similarity measure can be calculated by:

$$Leveshtein(s_1, s_2) = 1.0 - \frac{dist(s_1, s_2)}{max\ (|s_1|, |s_2|)}$$

where dist(s_1, s_2) refers to the actual Levenshtein distance function which returns a value of 0 if the strings are the same or a positive number of edits if they are different. The value of such a measure is between 0.0 and 1.0 where the bigger the value, the more similar the two strings.

(ii) *Smith-Waterman*

This algorithm was originally developed to find optimal alignment between biological sequences, like DNA or proteins. It is based on a dynamic programming

approach similar to Levenshtein distance, but allows gaps as well as character specific match scores [119]. Let t be the final best score obtained based on the dynamic programming matrix and g be the match score value. Here, Smith-Waterman similarity measure between two strings s_1 and s_2 is calculated by:

$$Smith - Waterman(s_1, s_2) = \frac{\mathrm{t}}{\min(|s_1|, |s_2|) \times \mathrm{g}}$$

(iii) *Jaro*

Jaro [120] introduced a string comparator that accounts for insertions, deletions and transpositions, which was mainly used for comparison of first and last names [13]. The basic Jaro algorithm is for two strings s_1, s_2:

1) *compute the string lengths.*
2) *find the number of common characters in the two strings.*
3) *find the number of transpositions.*

Given strings $s = s_1 \ldots s_k$ and $t = t_1 \ldots t_l$, define a character s_i in s to be common with t iff there is a $t_j = s_i$ in t such that $i\text{-}H \leq j \leq i+H$, where $H = \min(|s|, |t|)/2$. Let $s' = s'_1 \ldots s'_k$ be the characters in s which are common with t (in the same order they appear in s) and let $t' = t'_1 \ldots t'_l$ be the same in t. A transposition for s', t' is a position i such that $s'_i \neq t'_i$. Let $T_{s',t'}$ be half the number of transpositions for s' and t'. Jaro similarity measure for strings s and t is calculated by:

$$Jaro(s,t) = \frac{1}{3}\left(\frac{|s'|}{|s|} + \frac{|t'|}{|t|} + \frac{|s'| - T_{s',t'}}{|s'|} \right)$$

(iv) *Jaro-Winkler*

William Winkler proposed a variant of the Jaro metric based on empirical studies that fewer errors typically occur at the beginning of names [123]. Jaro-Winkler similarity measure between two string s_1 and s_2 is calculated by:

$$Jaro - Winkler(s_1, s_2) = Jaro(s_1, s_2) + \frac{p}{10}(1 - Jaro(s_1, s_2))$$

where *p* is the maximum number of the longest common prefix of two strings (s_1 and s_2) and is up to a maximum of 4 characters.

(v) *Q-Gram*

The Q-Gram metric is based on the intuition that two strings are similar if they share a large number of common q-grams. Q-grams are sub-strings of length q [122]. Commonly used q-grams are unigrams (q = 1), bigrams (q = 2) and trigrams (q = 3). For example, the bigrams for 'John' contains 'Jo', 'oh' and 'hn'. In this thesis, a q-gram similarity measure between two strings is calculated by counting the number of q-grams in common (i.e. q-grams contained in both strings) and divided by the maximum the number of q-grams in the two strings. Let $G_q(s)$ denote all the q-grams of a string s obtained by sliding a window of length q over the characters of s. The Q-Gram similarity measure between strings s_1 and s_2 is calculated by:

$$q - gram(s_1, s_2) = \frac{\left|G_q(s_1) \cap G_q(s_2)\right|}{max\left(\left|G_q(s_1)\right|, \left|G_q(s_2)\right|\right)}$$

5.4 Experiment and Experimental Results

Hassanzadeh *et al* [128] have undertaken a thorough comparison of token-level algorithms and claim that their accuracy highly depends on the characteristics of the data such as the amount and type of the errors and the length of strings. In our experiments, we also consider similar characteristics of the data but choose character-level algorithms for the test.

In this section, the experimental results on the performance of the selected five name matching techniques are presented. Especially, some recommendations proposed in table 5.1 are evaluated during the experiment. For example, the first recommendation from table 5.1 highlights the importance of different types of name

strings regarding the matching performance. Based on the experimental results, it has been noticed that the performance of a selected algorithms varies for different types of name strings (given name, surname, and full name). In our experiment, two types of strings are considered respectively, i.e., last name strings and first name strings.

Regarding the experiment on the last name strings, eight groups of datasets with different data sizes ranging from 200 records to 9454 records are carefully designed for the purpose of the experiments. In the experiment, the error rate of a dataset is defined as the ratio of erroneous records and the whole number of records in the dataset. There are three error rates considered for each group of datasets, i.e., low, medium and high with values of 20%, 50% and 70% respectively. For each size, three datasets with different error rates are used. For example, in the group of 9454 records of last name dataset, three datasets were designed:

a) *Low error rate 9454 records of last name dataset.*
b) *Medium error rate 9454 records of last name dataset.*
c) *High error rate 9454 records of last name dataset.*

With respect to the experiments for last name string, the following factors are considered:

- Effects of error rates on the selection of threshold values for different techniques
- Effects of error rates on the performance of different techniques
- Effects of sizes of datasets on the performance of different techniques
- Effects of error rates on the timing performance of different techniques

The recommendations proposed in table 5.1 will be evaluated based on the

experimental results. For example, in the fourth recommendation from table 5.1, the author proposes that Jaro and Winkler perform well for both given name strings and surname strings. However, since Christen's experimental work does not consider the different characteristics of a dataset as Hassanzadeh *et al*, it is unclear if this recommendation still holds considering the various characteristics of a dataset. Therefore, the fourth recommendation will be re-evaluated based on the experiments with different error rates and sizes of datasets. In addition, the issue of selecting a threshold value for an optimal matching quality proposed in the eighth recommendation in table 5.1 will be analyzed in detail.

Regarding the experiment on first name strings, it is expected that the experiments can help to answer following questions:

a) Whether the types of typos will result in a different accuracy performance of the selected algorithms. Three types of typos are considered in the proposed experiment: (i) a typo occurs in the front part of a given string (marked as TFP), (ii) a typo occurs in the latter part of a given string (marked as TLP), (iii) a typo occurs in any part of a given string randomly (marked as TR).

b) Whether the types of strings will result in a different performance of the selected algorithms. According to Christen, three types of name strings are used in the experimental work. It is noticed that the accuracy performance of the same algorithm will vary for the different types of name string values. This has been addressed in the first recommendation from table 5.1 exclusively. In the proposed experiments, regarding the types of strings, two types are considered for the five character-level algorithms (i.e., first name strings and last name strings). Experiments will be designed exclusively to compare the relative performance of each algorithm when different error rates of a dataset are concerned with the two types of name strings respectively.

In addition, the effects of error rate on the threshold values selection and the effects of error rates on the accuracy performance of the selected five algorithms are also considered in the experiments for first name datasets. There's only one size of 2300 records dataset for first name datasets. Similar to the experiments for the last name string values, the three error rates associated with the first name datasets are 20%, 50% and 70% respectively. It is expected that based on the experimental results, a recommendation will be made as to which technique should be selected in order to achieve the best matching quality when different name strings are considered under the different characteristics of a dataset.

5.4.1 Datasets preparation

In the absence of common datasets for data cleaning, we prepared our data for experiments as follows. With respect to the last name strings, the datasets are based on a historical set of real Electoral Roll data. First, a one million record dataset was extracted, from which a personal last name list was created. This list contains 9454 clean, non-duplicate personal last names. Then, a last name dataset was generated, which contains these 9454 last name records, with an ID number associated to each of the records. Erroneous records were created by doing the following four operations manually to the name field of records in the dataset: inserting, deleting, substituting and replacing characters. There were in total twenty-four datasets generated and the number of records for these last name datasets ranges from 200 records to 9454 records. Any last name dataset contained with the same records will have a different error rate associated with. Table 5.2 shows these datasets used for the last name experiments

Dataset	Error Rate		
9454 Records	Low	Medium	High
7154 Records	Low	Medium	High
5000 Records	Low	Medium	High
3600 Records	Low	Medium	High
2300 Records	Low	Medium	High
1000 Records	Low	Medium	High
500 Records	Low	Medium	High
200 Records	Low	Medium	High

Table 5.2 Datasets for last name experiments

With respect to the first name strings, 9 first name datasets were carefully designed, each of which contains 2300 records. These datasets are also based on a historical set of real Electoral Roll data. First, a one million record dataset was extracted, from which a personal first name list was created. This list contains 2300 clean, non-duplicate personal first names. Then, a first name dataset was generated, which contains these 2300 first name records, with an ID number associated to each of the records. Erroneous records were created applying the following four operations manually to the name field of records in the dataset: inserting, deleting, substituting and replacing characters. There are three different types of typos contained within the three groups of first name datasets, i.e., TFP, TLP, and TR. Any first name dataset contained with the same type of typo will have a different error rate associated with. Table 5.3 shows these first name datasets.

Dataset	Error Rate	Type of typo
2300 first name records	Low	TFP
2300 first name records	Low	TLP
2300 first name records	Low	TR
2300 first name records	Medium	TFP
2300 first name records	Medium	TLP
2300 first name records	Medium	TR
2300 first name records	High	TFP
2300 first name records	High	TLP
2300 first name records	High	TR

Table 5.3 First name datasets with different types of typos

For the purpose of comparing with the last name string values, 9 similar last name datasets are also designed. Any last name dataset containing the same type of typo will have a different error rate associated. Table 5.4 shows these last name datasets.

Dataset	Error Rate	Type of typo
2300 last name records	Low	TFP
2300 last name records	Low	TLP
2300 last name records	Low	TR
2300 last name records	Medium	TFP
2300 last name records	Medium	TLP
2300 last name records	Medium	TR
2300 last name records	High	TFP
2300 last name records	High	TLP
2300 last name records	High	TR

Table 5.4 Last name datasets with different types of typos

5.4.2 Measures

A target string is a *positive* if it is returned by a technique; otherwise it is a *negative*. A positive is a *true positive* if the match does in fact denote the same entity; otherwise it is a *false positive*. A negative is a *false negative* if the un-match does in fact denote the same entity; otherwise it is a *true negative*. The matching quality is evaluated using the F-measure (F) that is based on precision and recall:

$$F = \frac{2 \times P \times R}{P + R}$$

with P (precision) and R (recall) defined as:

$$P = \frac{|true\ positives|}{(|true\ positives| + |false\ positives|)}$$

$$R = \frac{|true\ positives|}{(|true\ positives| + |false\ negatives|)}$$

The most desirable algorithm is one that makes recall as high as possible without sacrificing precision. F-measure is a way of combining the recall and precision into a single measure of overall performance [130]. In the proposed experiments, precision, recall and F-measure are measured against different values of similarity threshold (i.e., θ). For the comparison of different techniques, the maximum F-measure scores across different threshold values are used.

5.4.3 Experimental results

5.4.3.1 Experimental results for Last name strings

(1) Effectiveness performance results

Regarding the effectiveness performance, the values of the maximum F-scores for

different algorithms on different last name datasets are presented in Appendix A (Table A.1). Figures Fig.5.2~Fig.5.9 present the corresponding results. For all graphs, the horizontal axis of the graph represents the techniques involved. The vertical axis of the graph represents the values of the maximum F-scores for different algorithms. Explanation for each figure is detailed below:

Fig.5.2 shows the maximum F-scores achieved by the five name matching algorithms on the 9454 last name datasets under three different error rates. It is clear to see in Fig.5.2 that the lower the error rate, the higher the maximum F-scores can be achieved for all the five name matching algorithms. When the error rate of a dataset is low, the effectiveness of Levenshtein, Jaro, and Q-Gram algorithms are equally the same followed by the Jaro-Winkler algorithm. The Smith-Waterman is the worst among the five algorithms. When error rate changes to medium, the Jaro algorithm becomes the best followed by the Jaro-Winkler algorithm. However when error rate is high, the Jaro-Winkler algorithm performs better than the Jaro algorithm. The Jaro-Winkler algorithm is the best among the five algorithms when the error rate of a dataset is high, followed by the the Levenshtein algorithm. The following table (table 5.5) shows the relative orders among the five algorithms regarding their maximum F-scores in the three different error rate datasets respectively.

Error rate	**Relative effectiveness order among the five algorithms**
Low	Levenshtein=Jaro=Q-Gram>Jaro-Winkler>Smith-Waterman
Medium	Jaro>Jaro-Winkler>Levenshtein>Q-Gram>Smith-Waterman
High	Jaro-Winkler>Levenshtein>Jaro>Q-Gram>Smith-Waterman

Table 5.5 Algorithms' order for 9454 last name dataset

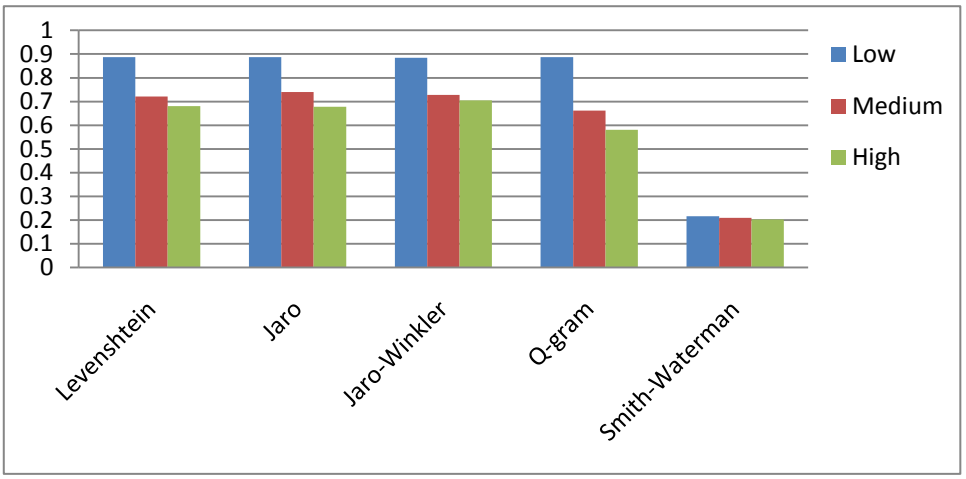


Fig.5.2 Effectiveness results for 9454 last name dataset

Fig.5.3 and Fig. 5.4 present the performances of these five name matching algorithms on the 7154 last name datasets and 5000 last name datasets respectively. The performances of these algorithms are exactly the same on both sizes datasets. Similarly to the performance on 9454 datasets, the lower the error rate, the higher the maximum F-scores are achieved for all the five name matching algorithms. When error rate is low, the relative performance of all five algorithms is exactly the same as they've performed on the 9454 last name dataset. When error rate of a dataset is changing to medium and high, the Jaro-Winkler algorithm becomes the best followed by the Jaro algorithm. The following table (table 5.6) shows the relative orders among the five algorithms regarding their maximum F-scores in the three different error rate datasets respectively.

Error rate	**Relative effectiveness order among the five algorithms**
Low	Levenshtein=Jaro=Q-Gram>Jaro-Winkler>Smith-Waterman
Medium	Jaro-Winkler>Jaro>Levenshtein>Q-Gram>Smith-Waterman
High	Jaro-Winkler>Jaro>Levenshtein>Q-Gram>Smith-Waterman

Table 5.6 Algorithms' order for 7154/5000 last name dataset

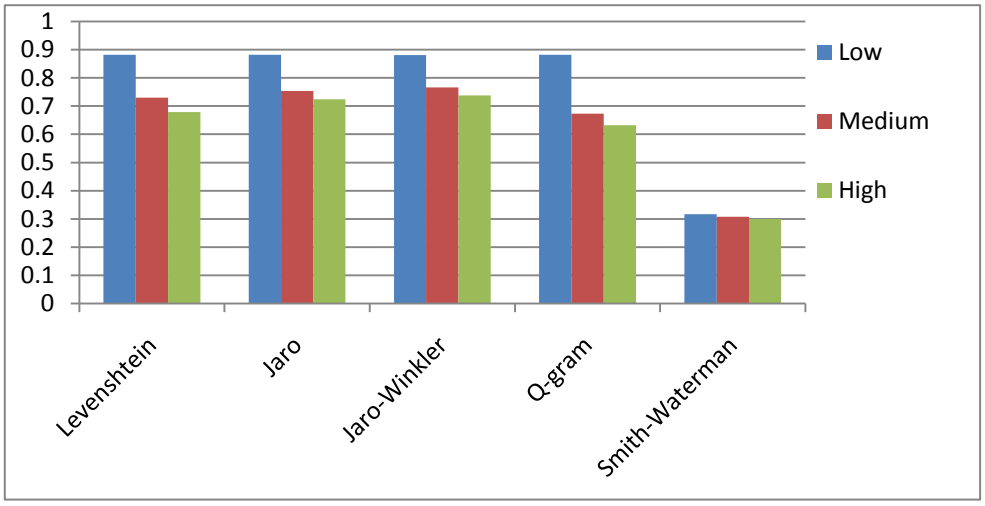


Fig.5.3 Effectiveness results for 7154 last name dataset

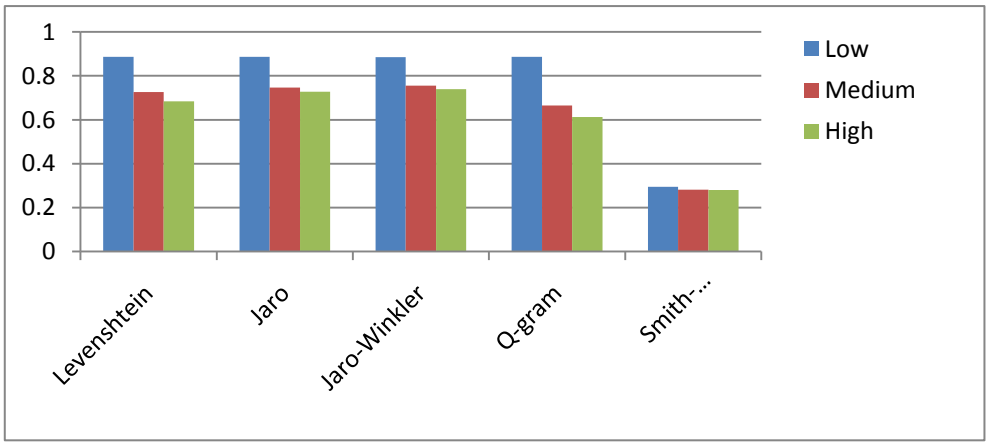


Fig.5.4 Effectiveness results for 5000 last name dataset

Fig.5.5 shows the performances of the five name matching algorithms on the 3600 last name datasets. Compared with their performances on the 7154 and 5000 last name datasets, the only differences are observed on the medium and high error rate datasets that the performance of the Jaro algorithm is the best instead of the Jaro-Winkler algorithm. The following table (table 5.7) shows the relative orders among the five algorithms regarding their relative maximum F-scores in the three different error rate datasets respectively.

Error rate	Relative effectiveness order among the five algorithms
Low	Levenshtein=Jaro=Q-Gram>Jaro-Winkler>Smith-Waterman
Medium	Jaro>Jaro-Winkler> Levenshtein>Q-Gram>Smith-Waterman
High	Jaro>Jaro-Winkler> Levenshtein>Q-Gram>Smith-Waterman

Table 5.7 Algorithms' order for 3600 last name dataset

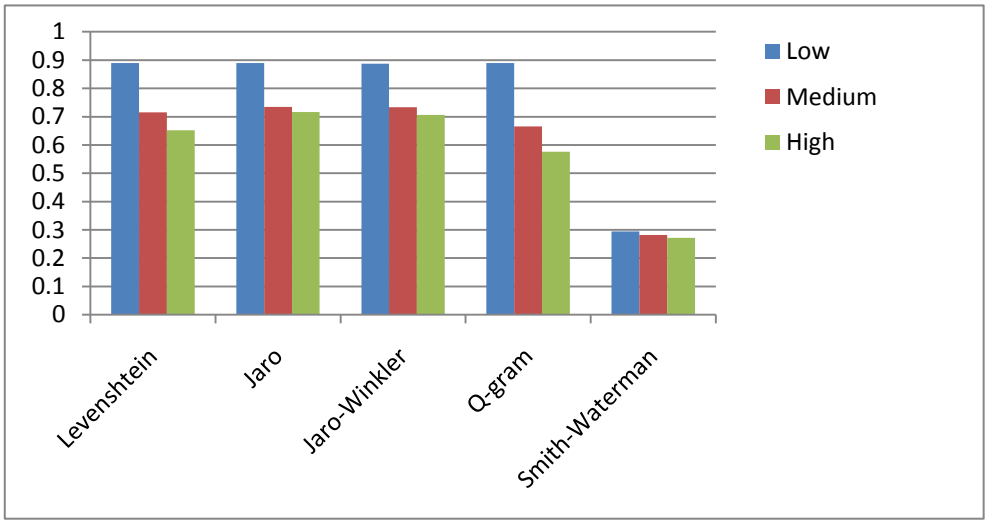


Fig.5.5 Effectiveness results for 3600 last name dataset

Fig.5.6 shows the performances of the five name matching algorithms on the 2300 last name datasets. These algorithms perform exactly the same as they have done on the 3600 last name datasets when error rate is low. When error rate changes to medium, the Jaro algorithm becomes the best followed by the Levenshtein algorithm. When error rate is high, the effectiveness performance of the Jaro-Winkler algorithm becomes the best among the five algorithms. The following table (table 5.8) shows the relative orders among the five algorithms regarding their maximum F-scores in the three different error rate datasets respectively.

Error rate	Relative effectiveness order among the five algorithms
Low	Levenshtein=Jaro=Q-Gram>Jaro-Winkler>Smith-Waterman
Medium	Jaro> Levenshtein>Jaro-Winkler> Q-Gram>Smith-Waterman
High	Jaro-Winkler> Jaro> Levenshtein>Q-Gram>Smith-Waterman

Table 5.8 Algorithms' order for 2300 last name dataset

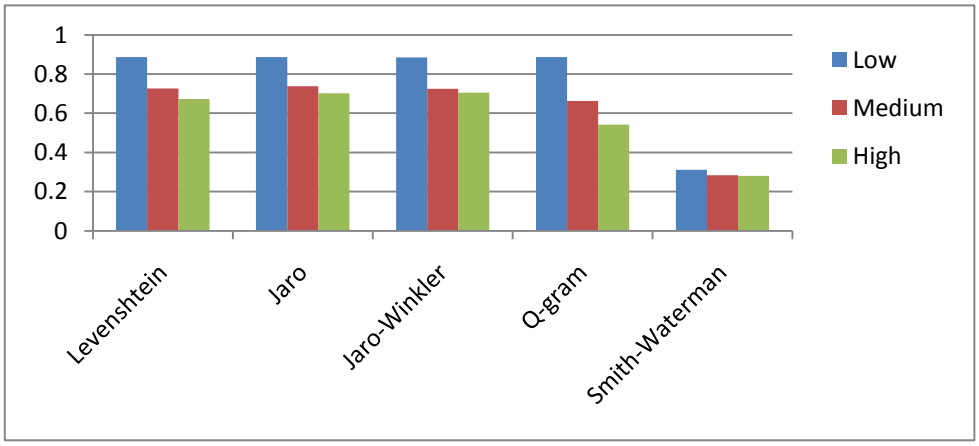


Fig.5.6 Effectiveness results for 2300 last name dataset

Fig.5.7 shows the performances of the five name matching algorithms on the 1000 last name datasets. Compared with 2300 last name datasets, their performances are exactly the same except that in high error rate dataset, the Levenshtein algorithm performs better than the Jaro algorithm. Table 5.9 shows the relative orders among the five algorithms regarding the maximum F scores achieved in the three different error rate datasets respectively.

Error rate	Relative effectiveness order among the five algorithms
Low	Levenshtein=Jaro=Q-Gram=Jaro-Winkler>Smith-Waterman
Medium	Jaro> Levenshtein>Jaro-Winkler> Q-Gram>Smith-Waterman
High	Jaro-Winkler> Levenshtein>Jaro> Q-Gram>Smith-Waterman

Table 5.9 Algorithms' order for 1000 last name dataset

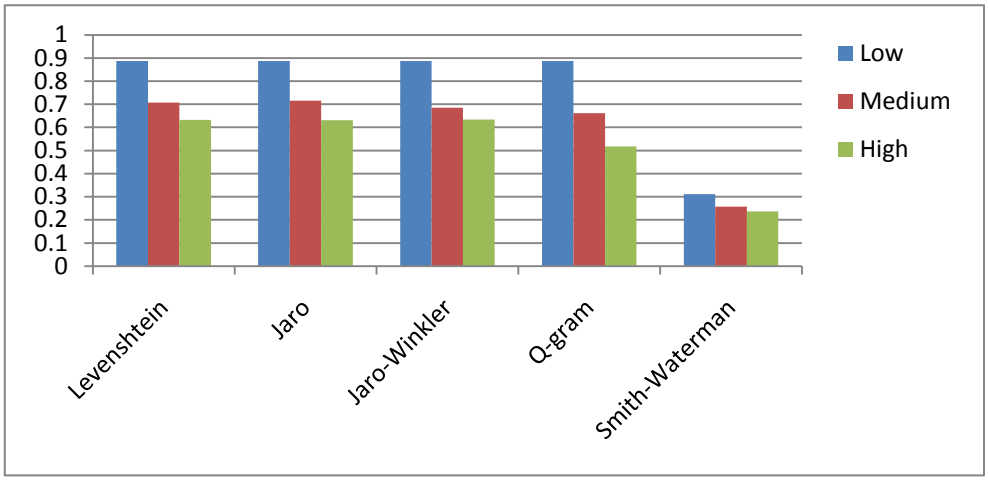


Fig.5.7 Effectiveness results for 1000 last name dataset

Fig.5.8 shows the performances of the five name matching algorithms on the 500 last name datasets. When error rate of a dataset is low, both the Levenshtein and the Jaro-Winkler algorithms are equally the best among the five algorithms, followed by the Jaro and the Q-Gram algorithms which perform equally the same. When the error rate is changing to medium, the Jaro-Winkler algorithm becomes the best followed by the Levenshtein algorithm. However, in high error rate dataset, the Jaro algorithm becomes the best among the five algorithms. The following table (table 5.10) shows the relative orders among the five algorithms regarding their maximum F-scores in the three different error rate datasets respectively.

Error rate	**Relative effectiveness order among the five algorithms**
Low	Levenshtein=Jaro-Winkler >Jaro=Q-Gram >Smith-Waterman
Medium	Jaro-Winkler> Levenshtein> Jaro> Q-Gram>Smith-Waterman
High	Jaro> Jaro-Winkler> Levenshtein> Q-Gram>Smith-Waterman

Table 5.10 Algorithms' order for 500 last name dataset

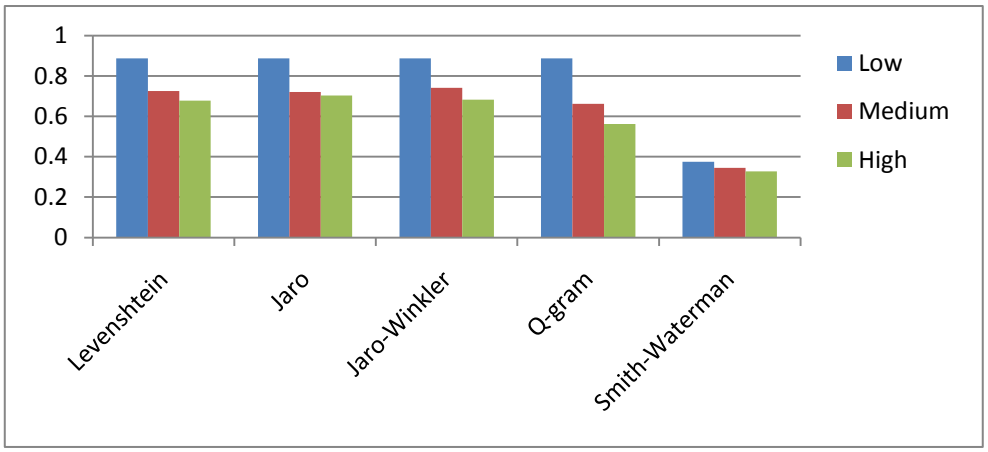


Fig.5.8 Effectiveness results for 500 last name dataset

Fig.5.9 shows the performances of the five name matching algorithms on the 200 last name datasets. It is clear to see in Fig.5.9 that apart from the Levenshtein algorithm, the lower the error rate, the higher the maximum F-scores can be achieved for the other four name matching algorithms. The maximum F-score for the Levenshtein algorithm in the low error rate dataset is the highest followed by its maximum F-score in the high error rate dataset rather than its maximum F-score in the medium error rate dataset. When error rate is low, the performances of Levenshtein, Jaro, Jaro-Winkler and Q-Gram algorithms are all equally the same. The Smith-Waterman's performance is the worst among the five algorithms. When the error rate is changing to medium or high, the Levenshtein algorithm becomes the best among the five algorithms. The following table (table 5.11) shows the relative orders among the five algorithms regarding their maximum F-scores in the three different error rate datasets respectively.

Error rate	**Relative effectiveness order among the five algorithms**
Low	Levenshtein=Jaro-Winkler =Jaro=Q-Gram >Smith-Waterman
Medium	Levenshtein> Q-Gram> Jaro> Jaro-Winkler>Smith-Waterman
High	Levenshtein> Q-Gram>Jaro> Jaro-Winkler> Smith-Waterman

Table 5.11 Algorithms' order for 200 last name dataset

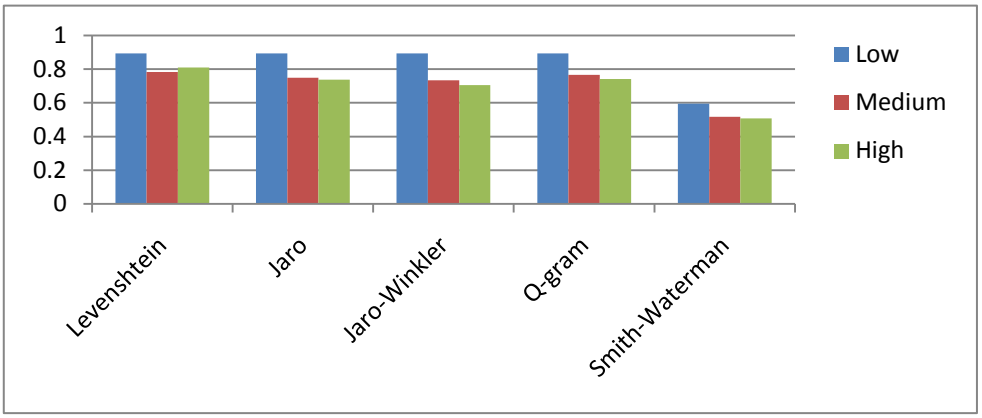


Fig.5.9 Effectiveness results for 200 last name dataset

Based on these experimental results regarding the effectiveness of the five name matching algorithms on different last name datasets, it can be concluded that generally these algorithms perform better in lower error rate datasets. In all last name datasets with low error rate, the Levenshtein algorithm remains one of the best effective algorithms among the five ones. In those medium or high error rate datasets, except for those datasets with 200 records, the Jaro or Jaro-Winkler algorithm remains the best choices. The Smith-Waterman algorithm however performs the worst among the five algorithms. Regarding the selection of a threshold value for each algorithm, values of thresholds of each algorithm obtaining the maximum F-scores in different last name datasets are shown in Appendix A (Table A.2). As shown in table A.2, generally, the higher the error rate of the dataset, the lower the threshold value should be chosen for an algorithm. For example, the three threshold values selected for the Levenshtein algorithm are 0.99, 0.85, 0.8 for low, medium, and high error rate dataset respectively when the size of a dataset is over 500 records. These experimental results achieved will be further analyzed later in section 5.5.

(2) Timing performance results

In general, the Jaro-Winkler algorithm requires the least running time among the five algorithms while the Smith-Waterman algorithm costs the most time. The time

requried by Jaro algorithm is slightly more than the time required by Jaro-Winkler algorithm. Regarding the timing performance, the Jaro algorithm and Jaro-Winkler algorithm are much better than the other three algorithms. The experimental results agree with that, the smaller size of a dataset, the lesser running time of an algorithm is required. According to the experimental results, the effect of error rate of a dataset on the timing performance for each algorithm is not significant. In Appendix A, table A.3 shows the average timing cost required by the five algorithms on the different sizes of datasets (9454, 7154, 5000, and 3600). The corresponding figures are shown in Fig.5.10, Fig.5.11, Fig.5.12, and Fig.5.13. For all graphs, the horizontal axis of the graph represents the algorithms involved. The vertical axis of the graph represents the timing cost in milliseconds. From these experimental results, it can be seen that for all datasets involved, the same order (i.e. Jaro-Winkler < Jaro < Levenshtein < Q-Gram < Smith-Waterman) from the least timing cost to the highest timing cost among the five algorithms is observed. Individually, in Fig.5.10, the higher the error rate of a dataset, the higher the timing cost is associated with an algorithm. This phenomenon is only observed for the Levenshtein and Q-Gram algorithms in Fig.5.11.

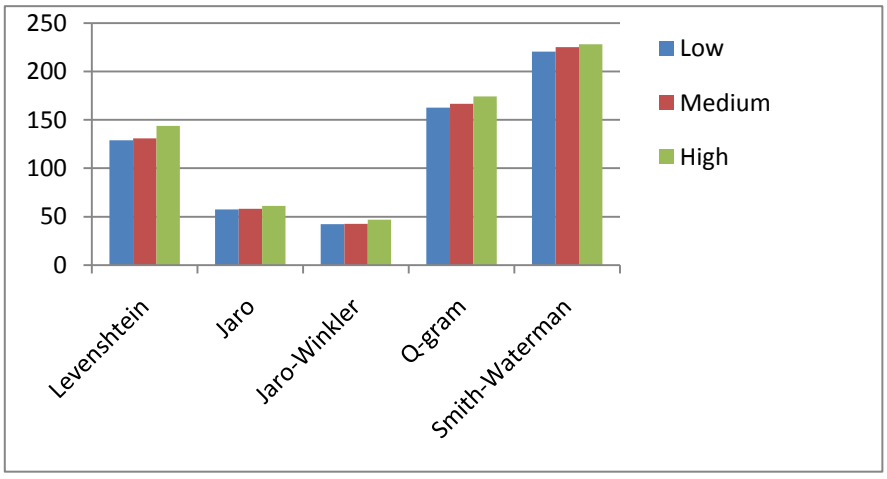


Fig.5.10 Timing performance in 9454 last name dataset

As shown in Fig.5.11, the least timing cost for the Jaro algorithm and Jaro-Winkler algorithm are observed for the medium error rate 7154 datasets, followed by the low error rate datasets. For the Smith-Waterman algorithm, the least timing cost is observed in the medium error rate 7154 dataset, followed by the high error rate dataset.

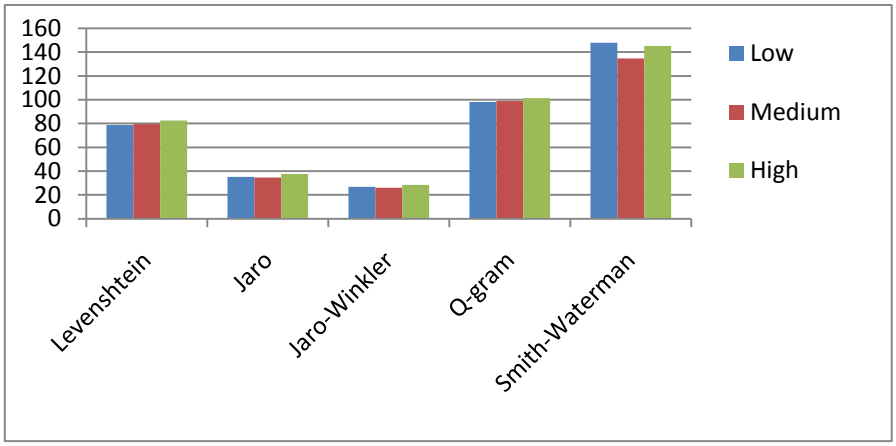


Fig.5.11 Timing performance in 7154 last name dataset

In Fig.5.12, except for the Levenshtein algorithm, the higher the error rate of a dataset, the higher the timing cost is required by the other four algorithms. The Levenshtein algorithm requires its most timing cost in medium error rate dataset.

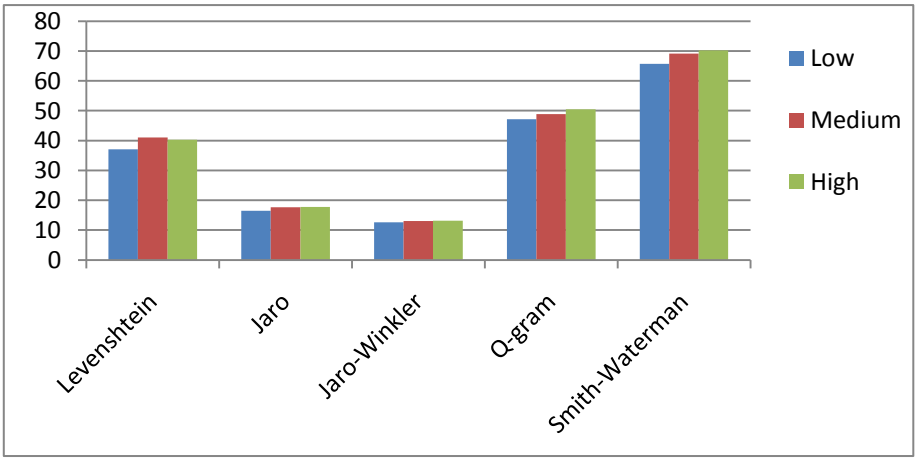


Fig.5.12 Timing performance in 5000 last name dataset

In Fig.5.13, except for the Smith-Waterman algorithm, the higher the error rate of a dataset, the higher the timing cost is required for the other four algorithms. The Smith-Waterman requires its most timing cost in medium error rate dataset

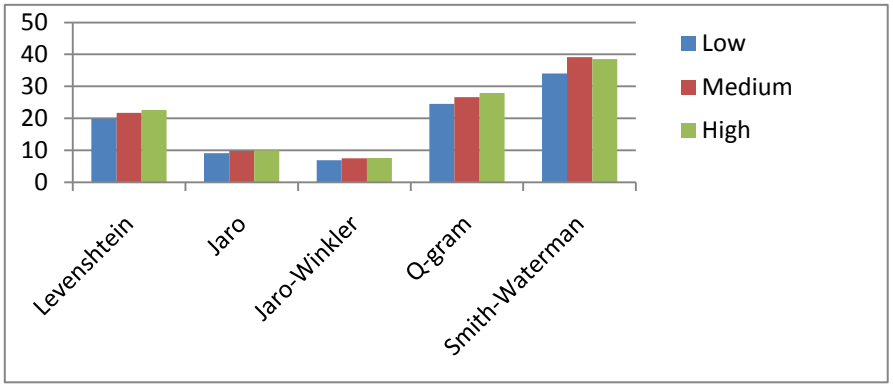


Fig.5.13 Timing performance in 3600 last name dataset

5.4.3.2 Experimental results for 2300 First name/ Last name strings

(1) Effectiveness performance results

The values of the maximum F-scores achieved by the five algorithms on different first name datasets with TFP, TLP, and TR typos are shown in Appendix A (Table A.4). The related graphs are generated and presented in Fig. 5.14~Fig.5.16. For all graphs, the horizontal axis of the graph represents the algorithms involved. The vertical axis of the graph represents the values of the maximum F-scores achieved by different algorithms.

In detail: Fig.5.14 shows the performances of the five name matching algorithms in the 2300 first name datasets with TFP typos under the three different error rates. It is clear to see in Fig.5.14 that in low error rate dataset, all algorithms performs better than the medium and high error rate datasets. The relative orders among the five algorithms regarding their maximum F-scores in the three different error rate

datasets are given in table 5.12. From table 5.12, it can be seen that the Levenshtein algorithm performs the best among the five algorithms in the datasets with TFP typos followed by Jaro algorithm. The Smith-Waterman algorithm is the worst among the five algorithms.

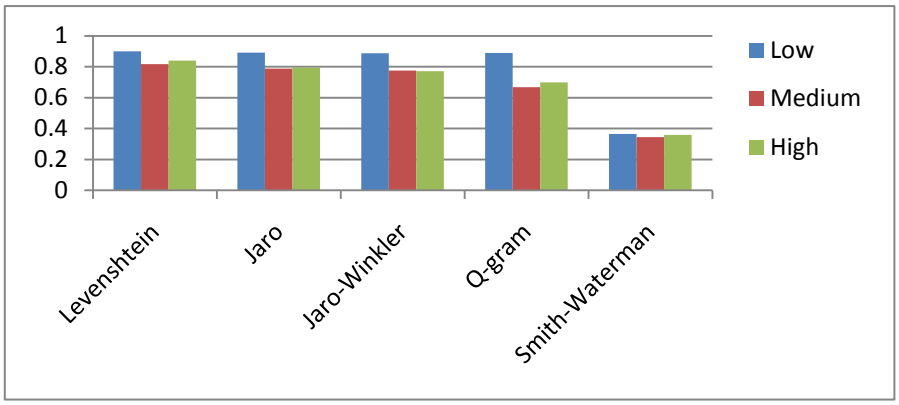


Fig.5.14 Effectiveness results for 2300 first name datasets with TFP typo

Error rate	**Relative effectiveness order among the five algorithms**
Low	Levenshtein>Jaro>Q-Gram >Jaro-Winkler>Smith-Waterman
Medium	Levenshtein>Jaro>Jaro-Winkler>Q-Gram>Smith-Waterman
High	Levenshtein>Jaro>Jaro-Winkler>Q-Gram>Smith-Waterman

Table 5.12 Algorithm's order for 2300 first name datasets with TFP typo

Fig.5.15 shows the performances of the five name matching algorithms in the 2300 first name datasets with TLP typos under the three different error rates. Their relative orders among the five algorithms regarding their maximum F-scores in the three different error rate datasets are given in table 5.13. It can be seen that unlike the first name datasets with TFP typos, the Levenshtein algorithm's performance is not the best among the five algorithms. The Jaro and Jaro-Winkler algorithms are the best choices in the three different error rate datasets.

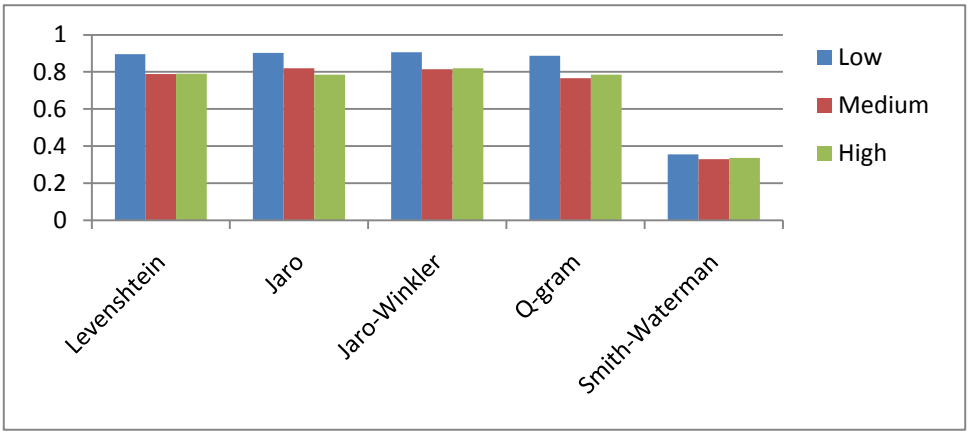


Fig.5.15 Effectiveness results for 2300 first name datasets with TLP typo

Error rate	**Relative effectiveness order among the five algorithms**
Low	Jaro-Winkler> Jaro> Levenshtein> Q-Gram >Smith-Waterman
Medium	Jaro>Jaro-Winkler>Levenshtein>Q-Gram>Smith-Waterman
High	Jaro-Winkler>Levenshtein>Q-Gram>Jaro>Smith-Waterman

Table 5.13 Algorithm's order for 2300 first name datasets with TLP typo

Fig.5.16 the performances of the five name matching algorithms in the 2300 first name datasets with TR typos under the three different error rates. In low error rate dataset, the Levenshtein algorithm performs the best among the five algorithms. In medium and high error rate datasets, the Jaro-Winkler and Jaro algorithms are the best choices respectively. Table 5.14 shows their relative orders regarding their maximum F-scores in the three different error rate datasets.

Error rate	**Relative effectiveness order among the five algorithms**
Low	Levenshtein>Jaro=Q-Gram >Jaro-Winkler>Smith-Waterman
Medium	Jaro-Winkler> Jaro>Levenshtein>Q-Gram>Smith-Waterman
High	Jaro>Jaro-Winkler>Levenshtein>Q-Gram> Smith-Waterman

Table 5.14 Algorithm's order for 2300 first name datasets with TR typo

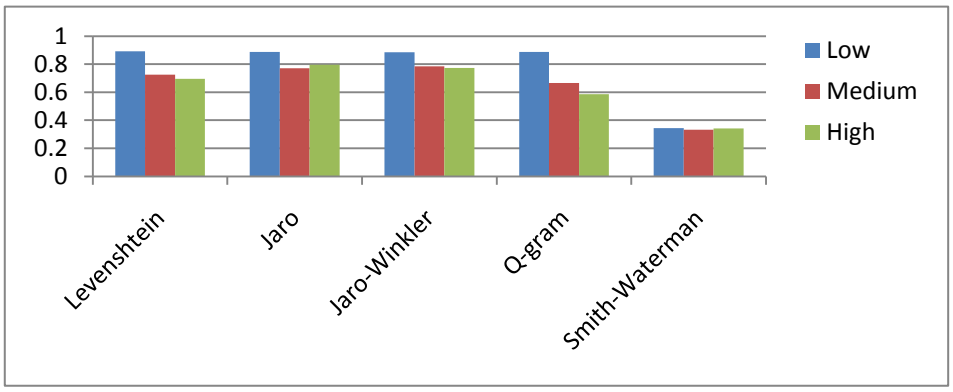


Fig.5.16 Accuracy results for 2300 first name datasets with TR typo

For the purpose of comparing with the 2300 last name datasets, these algorithms have been applied on 2300 last name datasets with the three different types of typos. The experimental results are achieved in Appendix A (Table A.5). Fig.5.17~Fig.5.19 are presenting the corresponding graphs. For all graphs, the horizontal axis of the graph represents the algorithms involved. The vertical axis of the graph represents the value of the maximum F-scores achieved by the five algorithms.

In detail: Fig.5.17 shows the performance of the five name matching algorithms in the 2300 last name datasets with TFP typos under three different error rates. All algorithms perform the best in low error rate datasets. Table 5.15 shows their relative orders regarding their maximum F-scores in the three different error rate datasets.

Error rate	**Relative effectiveness order among the five algorithms**
Low	Levenshtein>Jaro=Q-Gram >Jaro-Winkler>Smith-Waterman
Medium	Jaro> Levenshtein>Jaro-Winkler> Q-Gram>Smith-Waterman
High	Levenshtein>Jaro-Winkler Jaro> Q-Gram> Smith-Waterman

Table 5.15 Algorithm's order for 2300 last name datasets with TFP typo

Compared with the experimental results for the matching 2300 first name datasets, in the medium error rate dataset, the Jaro algorithm is the best choice rather than the Levenshtein algorithm.

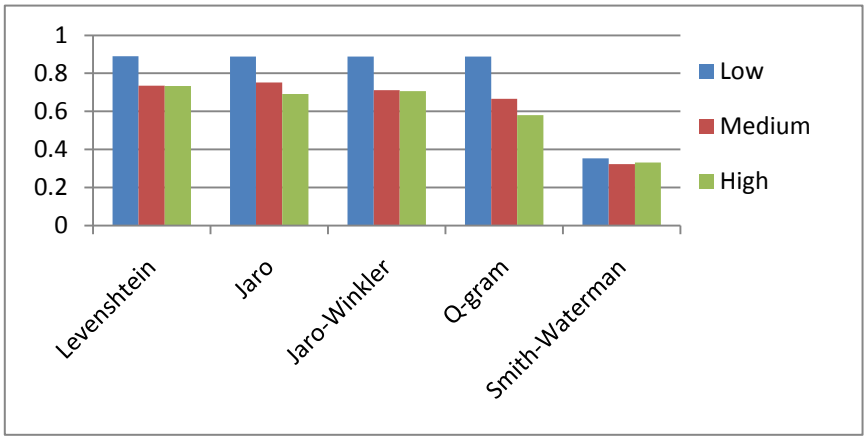


Fig.5.17 Accuracy results for 2300 last name datasets with TFP typo

Fig.5.18 shows the performances of the five name matching algorithms in the 2300 last name datasets with TLP typos. Compared with the matching 2300 first name datasets with TLP typos, performances of these algorithms are exactly the same in the medium and high error rate datasets (see table 5.16).

Error rate	**Relative effectiveness order among the five algorithms**
Low	Jaro-Winkler>Levenshtein>Jaro=Q-Gram >Smith-Waterman
Medium	Jaro>Jaro-Winkler> Levenshtein>Q-Gram>Smith-Waterman
High	Jaro-Winkler>Levenshtein>Jaro>Q-Gram>Smith-Waterman

Table 5.16 Algorithm's order for 2300 last name datasets with TLP typo

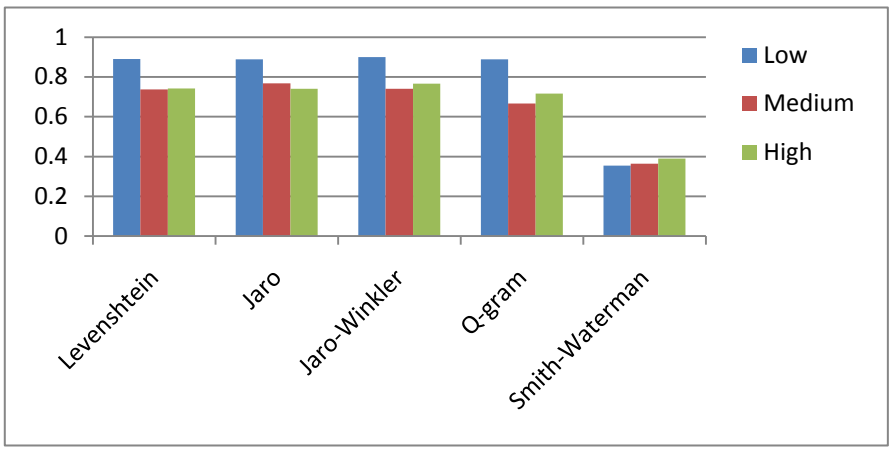


Fig.5.18 Accuracy results for 2300 last name datasets with TLP typo

Fig.5.19 shows the performance of the five name matching algorithms in the 2300 last name datasets with TR typos. According to their relative orders regarding their maximum F-scores in the three different error rate datasets (table 5.17), the best choices among the five algorithms compared with the matching 2300 first name datasets are totally different. For example, in medium error rate dataset, the Jaro algorithm is observed to perform the best while in first name datasets, the Jaro-Winkler algorithm is the best choice.

Error rate	**Relative effectiveness order among the five algorithms**
Low	Levenshtein=Jaro=Q-Gram>Jaro-Winkler>Smith-Waterman
Medium	Jaro>Levenshtein>Jaro-Winkler>Q-Gram>Smith-Waterman
High	Jaro-Winkler>Jaro>Levenshtein>Q-Gram>Smith-Waterman

Table 5.17 Algorithm's order for 2300 last name datasets with TR typo

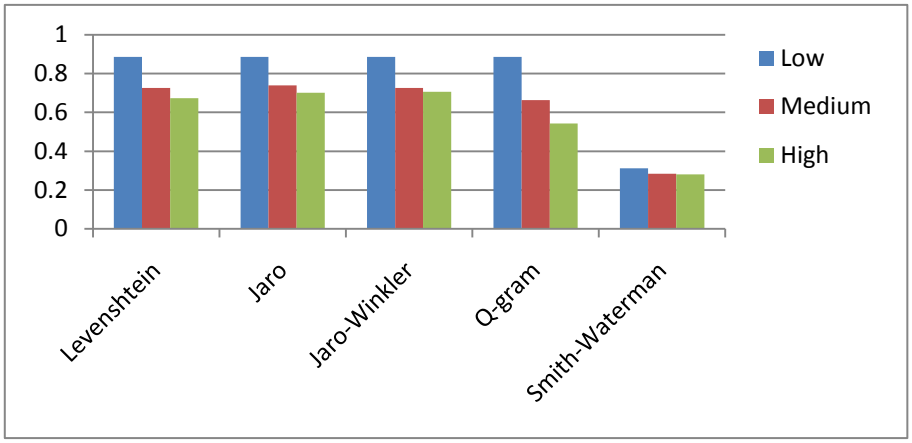


Fig.5.19 Accuracy results for 2300 last name datasets with TR typo

Table 5.18~Table 5.20 show the different threshold values selected for each algorithm to obtain the maximum F-scores in the different 2300 first/last name datasets with TFP, TLP, and TR typos respectively.

String Type	Algorithm	Low	Medium	High	Data Size	Error Typo
First name	Levenshtein	0.9	0.8	0.8	2300	TFP
First name	Jaro	0.95	0.95	0.9	2300	TFP
First name	Jaro-Winkler	0.99	0.95	0.95	2300	TFP
First name	Q-Gram	0.99	0.99	0.75	2300	TFP
First name	Smith-Waterman	0.99	0.99	0.99	2300	TFP
Last name	Levenshtein	0.9	0.85	0.8	2300	TFP
Last name	Jaro	0.99	0.95	0.95	2300	TFP
Last name	Jaro-Winkler	0.99	0.95	0.95	2300	TFP
Last name	Q-Gram	0.99	0.99	0.8	2300	TFP
Last name	Smith-Waterman	0.99	0.99	0.99	2300	TFP

Table 5.18 Threshold value selection for first/last name dataset with TFP typos

String Type	Algorithm	Low	Medium	High	Data Size	Error Typo
First name	Levenshtein	0.9	0.8	0.8	2300	TLP
First name	Jaro	0.95	0.95	0.9	2300	TLP
First name	Jaro-Winkler	0.99	0.95	0.95	2300	TLP
First name	Q-Gram	0.99	0.85	0.8	2300	TLP
First name	Smith-Waterman	0.99	0.99	0.99	2300	TLP
Last name	Levenshtein	0.9	0.85	0.8	2300	TLP
Last name	Jaro	0.99	0.95	0.95	2300	TLP
Last name	Jaro-Winkler	0.99	0.95	0.95	2300	TLP
Last name	Q-Gram	0.99	0.99	0.8	2300	TLP
Last name	Smith-Waterman	0.99	0.99	0.99	2300	TLP

Table 5.19 Threshold value selection for first/last name dataset with TLP typos

String Type	Algorithm	Low	Medium	High	Data Size	Error Typo
First name	Levenshtein	0.9	0.8	0.8	2300	TR
First name	Jaro	0.99	0.9	0.9	2300	TR
First name	Jaro-Winkler	0.99	0.95	0.95	2300	TR
First name	Q-Gram	0.99	0.99	0.75	2300	TR
First name	Smith-Waterman	0.9	0.85	0.85	2300	TR
Last name	Levenshtein	0.99	0.85	0.8	2300	TR
Last name	Jaro	0.99	0.95	0.9	2300	TR
Last name	Jaro-Winkler	0.99	0.95	0.95	2300	TR
Last name	Q-Gram	0.99	0.99	0.75	2300	TR
Last name	Smith-Waterman	0.9	0.9	0.9	2300	TR

Table 5.20 Threshold value selection for first/last name dataset with TR typos

It can be deduced from the experimental data obtained in table 5.18~table 5.20 that characteristics such as the types of string, types of typos, error rate may influence the selection of a proper threshold value for the selected algorithm to achieve the best effectiveness performance. They will be further evaluated in section 5.5 in detail .

(2) Timing performance results

The timing performance of the five algorithms in these first name datasets are exactly the same as they performed in those last name datasets, i.e., the Jaro-Winkler

costs least running time among the five algorithms while the Smith-Waterman costs the most running time. The running time required by Jaro algorithm is slightly more than Jaro-Winkler algorithm. Both algorithms perform better than the other three algorithms. Experimental results show that the effect of error rate of a dataset is not significant on the timing performance.

5.5 Evaluation

5.5.1 Last name experimental results evaluation

The test results for last name datasets will be evaluated and analyzed based on the effectiveness and timing performance of the five selected techniques.

Similar experiments have been done on last name datasets with records ranging from 200 to 9454 respectively, and the results show that in general, the size of a dataset is not sensitive to the effectiveness relative to the threshold values when it is above 1000 records, except for Smith-Waterman.

When the size of a dataset is smaller than 1000 records, the best F-score is relative to the value of thresholds on different datasets with different error rates. The corresponding experimental results are given in table A.2 in Appendix A.

Particularly, figures in appendix A (Fig.A.1~Fig.A.5) represent the results of the effectiveness relative to the values of threshold on the size of 3600 last name datasets with different error rates for the five algorithms. For all graphs, the horizontal axis is the values of threshold. According to the experimental results, the following results are achieved:

1) Effect of Error Rates on Threshold Values:

As shown in table A.2 from Appendix A, generally for all techniques, the higher the error rate in the dataset, the lower the threshold value is required in order to achieve the best effectiveness performance. For example, the Jaro algorithm performs the best in high error rate dataset at threshold value of 0.9, while it performs the best over the datasets with medium and low error rate at threshold values of 0.95 and 0.99 respectively when the size of a dataset is above 1000. It is recommended that for algorithms like Levenshtein, Jaro, Jaro-Winkler and Q-Gram, the higher the error rate, the lower the threshold value should be selected.

2) Effect of the Sizes of Datasets on Threshold value:

Table A.2 presents the different threshold values selected for each algorithm in the 8 groups of last name datasets. In general the selected threshold value is not sensitive to the size of a dataset except for the Smith-Waterman algorithm. For example, the three selected threshold values for the Jaro algorithm in the last name dataset with 1000 records are 0.99, 0.95 and 0.9 for low error rate, medium error rate and high error rate datasets respectively. These three threshold values remain the same with increasing the sizes of datasets up to 9454 records. However, for Smith-Waterman algorithm, it is noticed that the selection of a threshold value is quite sensitive to the size of a dataset. For example, in high error rate datasets, the threshold value selected for Smith-Waterman algorithm in dataset with 9454 records is 0.9, while the value is changed to 0.99 in dataset with of 7154 records and the threshold value changes back to 0.9 again in the dataset with 5000 records.

3) Effect of Error Rates on Effectiveness Performance:

For all eight groups of last name datasets, experimental results show that in general,

all five algorithms perform better in low error rate datasets. For example, Fig. 5.20 shows performance of the five algorithms on datasets with 3600 records under the three different error rates. It is observed that the performance of the five algorithms is decreasing along with the increasing error rate. Only one exception is noticed in the datasets with 200 records where the performance for the Levenshtein algorithm in a high error rate dataset is higher than that in the medium error rate dataset. Fig.5.21 shows that in the last name dataset with 200 records, the Levenshtein algorithm performs the best in low error rate dataset followed by the high error rate dataset.

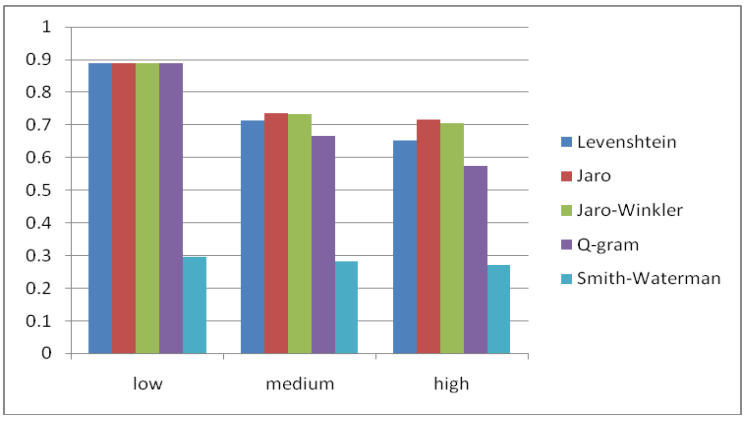


Fig.5.20 Maximum F score comparison on last name datasets size 3600

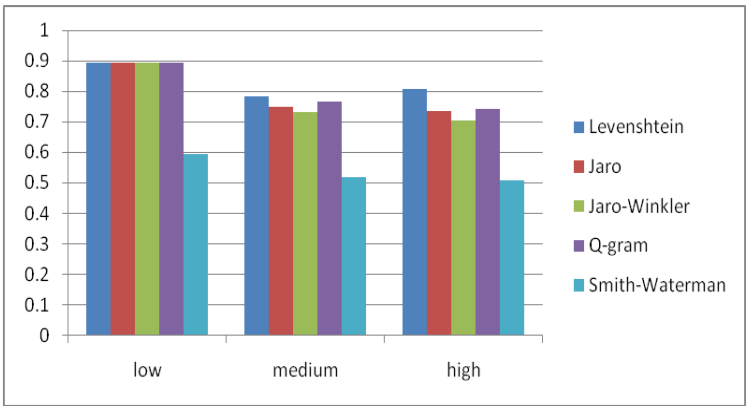


Fig.5.21 Maximum F score comparison on last name datasets size 200

Looking at the individual algorithm's performance, it was observed that the Jaro and Jaro-Winkler algorithms do not always perform the best for last name strings as suggested by Christen [124]. In the 24 last name datasets prepared for the experiments, only 9 datasets are observed that both the Jaro and Jaro-Winkler algorithms perform well for last name strings. On the contrary, in most of low error rate datasets, the Jaro-Winkler algorithm ranked at the fourth position among the five algorithms. Thus it is recommended that when selecting a suitable algorithm for the matching of names, the characteristics of a dataset such as the *dataset size*, *data error rate* should be considered.

4) Effect of the Sizes of Datasets on Effectiveness Performance:

By comparing the relative performance among the five algorithms in table A.1, the effectiveness performance of the algorithms are analyzed in last name datasets under the three different error rates. The results are listed in the following tables (table 5.21~table 5.23):

Dataset	**Relative effectiveness order among the five algorithms**
9454	Levenshtein=Jaro=Q-Gram>Jaro-Winkler>Smith-Waterman
7154	Levenshtein=Jaro=Q-Gram>Jaro-Winkler>Smith-Waterman
5000	Levenshtein=Jaro=Q-Gram>Jaro-Winkler>Smith-Waterman
3600	Levenshtein=Jaro=Q-Gram>Jaro-Winkler>Smith-Waterman
2300	Levenshtein=Jaro=Q-Gram>Jaro-Winkler>Smith-Waterman
1000	Levenshtein=Jaro=Q-Gram>Jaro-Winkler>Smith-Waterman
500	Levenshtein=Jaro-Winkler>Jaro=Q-Gram>Smith-Waterman
200	Levenshtein=Jaro-Winkler=Jaro=Q-Gram>Smith-Waterman

Table 5.21 Algorithms' order for low error rate last name datasets

Dataset	Relative effectiveness order among the five algorithms
9454	Jaro >Jaro-Winkler> Levenshtein>Q-Gram>Smith-Waterman
7154	Jaro-Winkler> Jaro > Levenshtein>Q-Gram>Smith-Waterman
5000	Jaro-Winkler> Jaro > Levenshtein>Q-Gram>Smith-Waterman
3600	Jaro >Jaro-Winkler> Levenshtein>Q-Gram>Smith-Waterman
2300	Jaro > Levenshtein>Jaro-Winkler> Q-Gram>Smith-Waterman
1000	Jaro > Levenshtein>Jaro-Winkler> Q-Gram>Smith-Waterman
500	Jaro-Winkler> Levenshtein> Jaro > Q-Gram>Smith-Waterman
200	Levenshtein> Q-Gram> Jaro > Jaro-Winkler> Smith-Waterman

Table 5.22 Algorithms' order for medium error rate last name datasets

Dataset	Relative effectiveness order among the five algorithms
9454	Jaro-Winkler> Levenshtein> Jaro > Q-Gram>Smith-Waterman
7154	Jaro-Winkler> Jaro > Levenshtein> Q-Gram>Smith-Waterman
5000	Jaro-Winkler> Jaro > Levenshtein> Q-Gram>Smith-Waterman
3600	Jaro > Jaro-Winkler> Levenshtein> Q-Gram>Smith-Waterman
2300	Jaro-Winkler> Jaro > Levenshtein> Q-Gram>Smith-Waterman
1000	Jaro-Winkler> Levenshtein> Jaro > Q-Gram>Smith-Waterman
500	Jaro > Jaro-Winkler> Levenshtein> Q-Gram>Smith-Waterman
200	Levenshtein> Q-Gram> Jaro > Jaro-Winkler> Smith-Waterman

Table 5.23 Algorithms' order for high error rate last name datasets

Table 5.22 shows that in the low error rate dataset, the algorithm with the best performance among the five algorithms are not quite as sensitive to the size of a dataset. For low error rate datasets with records over 1000, Levenshtein, Jaro, and Q-Gram perform equally well. However, in medium error rate datasets, the best choice varies between Jaro algorithm and Jaro-Winkler until the size of the dataset

becomes 200. This is also observed in high error rate datasets as presented in table 5.23.

5.5.2 2300 first/last name experimental result evaluation

In section 5.4.3.2, experimental results on a group of 2300 first name datasets as well as a group of 2300 last name datasets are presented. In this section, these experimental results will be evaluated and analyzed based on six aspects, which will be detailed below:

1) The Effect of Error Rates on Threshold Values:

Experimental results for 2300 first name datasets show that in general, the higher the error rate of the dataset, the lower the threshold value should be selected expect for the Smith-Waterman algorithm. For example, in high error rate first name dataset with TLP typos, the Q-Gram algorithm achieves the best F-score with the selected threshold value of 0.8, while it achieves the best F-score with the selected threshold values of 0.85 and 0.99 in medium error rate dataset and low error rate dataset respectively. Table 5.18~Table 5.20 shows the results of the selected threshold values for all five algorithms achieving the maximum F-scores in the different 2300 first name datasets with TFP, TLP, and TR typos respectively. According to these tables, it is clear to see that for all first name datasets containing TFP and TLP typos, the threshold values selected for the Smith-Waterman algorithm remains the same under the three different error rates.

2) The Effect of Error Rates on Effectiveness Performance:

Experimental results from appendix A.4 show that in general, all five algorithms perform best in low error rate first name datasets (see table A.4). Looking at the performance of individual algorithm, it is observed that in all 2300 first name

datasets with TFP typos, the Levenshtein algorithm appears to be the best effective algorithm among the five algorithms no matter what error rate is associated in the dataset (see Fig.5.22). For other first name datasets with TLP typos and TR typos, the best effective algorithm is sensitive to the error rate of the dataset. Generally, either the Jaro or Jaro-Winkler algorithm should be selected in order to achieve the best matching quality according to the different error rate.

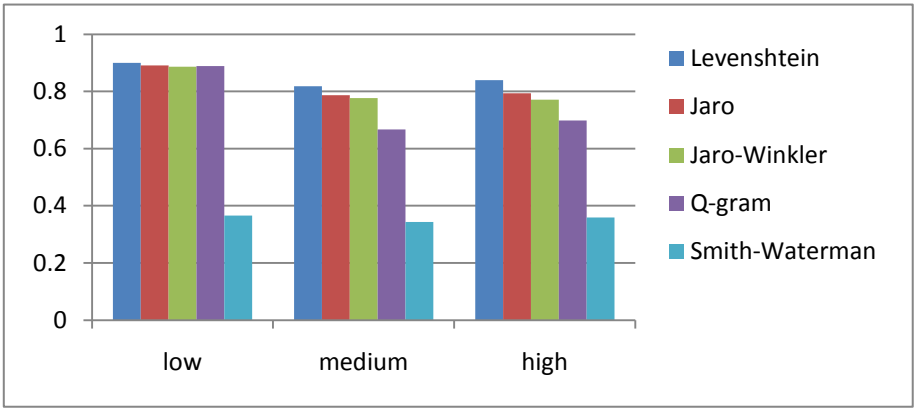


Fig.5.22 Performance comparisons on 2300 first name datasets with TFP typo

3) The Effect of Types of Typos on Threshold Values

With respect to the first name datasets, the Levenshtein and Jaro-Winkler algorithms are both not sensitive to the types of typos regarding the selection of the threshold values according to the experimental results. The other three algorithms require the changing of the threshold values in order to achieve the best F-scores in these first name datasets with different types of typos. With respect to the last name datasets, it is noticed that only the Jaro-Winkler algorithm is not sensitive to the types of typos regarding the selection of the threshold values. The three threshold values selected for the low error rate, medium error rate, and high error rate last name datasets are 0.99, 0.95, and 0.95 respectively no matter what types of typos are involved in the datasets.

4) The Effect of Types of Strings on Threshold values

For datasets containing the same type of typos, the experimental results are evaluated on the first name strings and last name strings respectively. It is discovered that for some algorithms, when different type of strings are involved, the proper threshold value for different algorithm may be varied. For example, in the datasets containing only TFP typos, the threshold value selected for Jaro algorithm in low error rate first name dataset is 0.95 and it is required to be increased to 0.99 in order to achieve the best effectiveness performance. Only the Jaro-Winkler and Smith-Waterman algorithms are not sensitive to the different types of strings involved in the datasets containing only TFP typos.

5) The Effect of Types of Strings on Effectiveness Performance

According to the experimental results presented in appendix A.4 and A.5, in general, algorithms perform better on first name strings than last name strings. It is estimated that the reason might be due to the length of the string varies between the two types of strings. Fig.5.23 shows the relative performance between first name strings and last name strings under the three different error rates. The types of typos contained in these datasets are all TFP typos. It is noticed that the types of strings will influence the performance. For example, in medium error rate datasets, the Levenshtein algorithm performs the best on first name strings while the Jaro algorithm is the best on last name strings. It is estimated that the different performance is due to the different length of the selected strings, though further experiments regarding the different length of the name strings have not been undertaken.

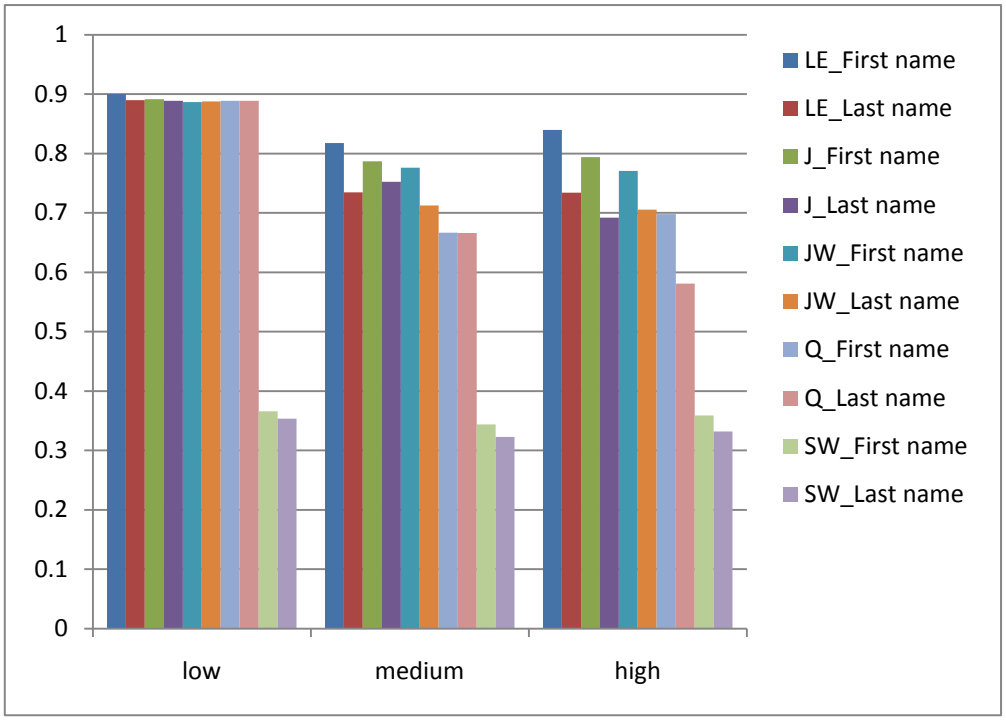


Fig.5.23 Performance comparisons between first name datasets and last name datasets with TFP typos under three different error rates

5.6 Summary

Based on the evaluation results discussed in section 5.5, the following recommendations are made:

(1) Regarding the threshold value selection, the error rate of a dataset, the types of strings involved and the types of typos in the string will all influence the selection of a suitable threshold value for the selected algorithm in order to achieve the best effectiveness performance. However, the selected threshold values are not sensitive to the changes of the size of a dataset. It is recommended that the higher the error rate, the lower the threshold value should be chosen. With the help of the experimental results achieved in this chapter, table A.6 in appendix A.11 presents a list of suggestions regarding the selection of a suitable algorithm as well as the selection of the required threshold values considering the different characteristics of

a dataset.

(2) Regarding the effectiveness performance of a selected algorithm, the error rate, the types of strings will all influence the effectiveness performance of the selected algorithm. In general, the algorithms perform better in lower error rate datasets. Between the first name datasets and last name datasets, an algorithm performs better in first name datasets.

(3) For names parsed into separate fields, the Jaro and Jaro-Winkler algorithms are not always among the best choices for matching the first name strings or last name strings. The best choice regarding the effectiveness of name matching algorithms involves concerning the error rate, size, and types of strings associated with a dataset.

(4) If speed is important, algorithms such as Jaro, Jaro-Winkler should be selected. The Smith-Waterman algorithm should not be selected for the purposed of matching name strings parsed into separate fields.

5.7 Conclusion

This chapter has analyzed and evaluated five popular character-based name matching techniques. A comprehensive comparison of the five techniques has been done based on a series of experiments on different last name and first name datasets. The experimental results confirmed the statement that there is no clear best technique. The size of dataset, the error rate in a dataset, the types of strings in a dataset and the types of typos in a string will all affect on the performance of a selected algorithm.

Regarding the threshold value selection, according to the experimental results, the

higher the error rate in the dataset, the lower the threshold value required in order to achieve the best performance. Timing performance based on the five algorithms on different datasets has also been analyzed and compared. Overall, Jaro-Winkler and Jaro are significantly faster than others.

With the help of the achieved experimental results in this chapter, recommendations on the selection of a suitable algorithm for a particular name matching task are proposed in section 5.6. Compared with the existing recommendations addressed in the previous research, ours provide a group of much more detailed recommendations with the corresponding parameters supplied for the recommended algorithms for practical use.

In detail, four recommendations proposed by Christen are further evaluated in this chapter. Christen claimed that the type of string should be considered for a selection of a suitable algorithm without giving any further suggestions based on the algorithms used in his work. In this chapter, two types of strings (last name strings and first name strings) are used for the evaluation respectively and proposed a detailed selection of algorithms according to the different characteristics associated within a dataset (see table A.6). According to Christen, it is claimed that the Jaro and Jaro-Winkler algorithms seem to perform well for both first name strings and last name strings and are recommended to be selected during the matching task of the personal name strings. However, this recommendation does not always hold according to the experimental results in this chapter. For example, in high error rate last name dataset with 200 records, the Levenshtein algorithm is among the most accurate measures followed by the Q-Gram algorithms. In this case, Jaro and Jaro-Winkler however are not suitable for the matching task.

Regarding the threshold value selection problem for an algorithm, Christen highlighted the difficulty in setting an optimal threshold value and claimed that an

optimal threshold value vary between different datasets without giving any further evaluation regarding the threshold value selection problem. In the proposed experiments, a thorough evaluation regarding the selection of a suitable threshold value for different algorithms is made and detailed threshold values are presented with different characteristics of data concerned. Based on these data values, a similar regular pattern is observed compared with that of Hassanzadeh *et al*, i.e., a lower value of the threshold is needed as the error rate in the dataset increases.

All algorithms selected in this chapter are character-level algorithms rather than the token-level algorithms used by Hassanzadeh *et al* [128]. Hassanzadeh *et al* focus the evaluations regarding the effectiveness of the different token-level algorithms mainly on two characteristics: the error rate of a dataset and the type of errors. Compared with Hassanzadeh *et al*, the proposed evaluations in this chapter addressed more characteristics: the error rate of a datasets, the type of string, the size of dataset, the type of typo. All these characteristics are used during the evaluation of the relative effectiveness of the five algorithms as well as the selection of a threshold value. However, in the proposed experiments, only one error type is considered (misspelling) and while Hassanzadeh *et al* involve three types of errors (misspelling, abbreviation and word swap).

CHAPTER 6 CONCLUSION AND FUTURE WORK

The main outcomes of the research undertaken for this thesis are the development of a rule-based taxonomy of dirty data, a novel data cleaning framework, and the evaluation work towards the performance of five popular approximate string matching algorithms. This chapter discusses two aspects of the work that merit further examination and discussion. Firstly, the conclusions and contributions are discussed and summarized. Secondly, the future directions of the research are discussed.

6.1 Novelties and contributions

- A rule-based taxonomy of dirty data

Today, data has become more and more important, with many human activities relying on it. As data have kept increasing at an explosive rate, a great number of database applications have been developed in order to derive useful information from these large quantities of data, such as decision support systems and customer relationship management systems (CRM). It has now been recognized that an inordinate proportion of data in most data sources is dirty.

Due to the 'garbage in, garbage out' principle, dirty data will distort information obtained from it. Obviously, a database application such as a data warehouse with a high proportion of dirty data is not reliable for the purpose of data mining or deriving business intelligence and the quality of decisions made on the basis of such business intelligence is also not convincing. Therefore, before using these database applications, dirty data needs to be cleaned. Due to the lack of appreciation of the types and extent of dirty data in many enterprises, inadequate attention is paid to the

existence of dirty data in many database applications. Besides, methodologies are not applied to ensure high quality data in these applications.

In this thesis, research work regarding the studies of dirty data types is reviewed firstly. 38 dirty data types are proposed with the help of the studies of different data quality rules. Comparing with the dirty data types mentioned by the previous researchers, the proposed 38 dirty data types is the most complete collection of dirty data types. Although it is not ensured that all possible dirty data types that may exist are covered within the collection of these 38 dirty data types, it is believed that most usual or unusual dirty data types are included. Secondly, a rule-based taxonomy of dirty data is proposed based on these 38 dirty data types. The rule-based taxonomy of dirty data is introduced by associating the proposed 38 dirty data types under different data quality rules, which forms an even larger collection of dirty data compared with any of the existing taxonomies or classifications. With the help of the taxonomy, a method to deal with the DDS problem is developed by prioritizing the expensive process of data cleaning. By using the proposed rule based taxonomy during the data cleaning process, the business enterprises are maximally benefited.

- A novel data cleaning framework

In this thesis, a novel data cleaning framework has been proposed, which aims to challenge the following issues: (i) minimising the data cleaning time and improving the degree of automation in data cleaning, (ii) improving the effectiveness of data cleaning. Additionally, the proposed framework offers a function (The DDS process) to address some special cases when individual business requirements are involved. This function can help a business to take into account the special needs according to different businesses priority policies.

The proposed framework retains the most appealing characteristics of existing data

cleaning approaches, and improves the efficiency and effectiveness during a data cleaning process. Compared with existing data cleaning approaches, the proposed framework provides several exclusive features which have not been addressed in existing approaches.

Firstly, the proposed framework tries to address as many dirty data types as possible according to the proposed taxonomy of dirty data. Existing approaches only focus on specific data cleaning tasks such as data standardization or duplicate records elimination. Some tool only focuses on solving one activity such as ARKTOS. According to the knowledge to the author, none of the existing tools can help with providing an all-in-one solution to the problems mentioned in the proposed dirty data taxonomy.

Secondly, the proposed framework addresses the order of various cleaning activities exclusively and provides an automatic solution to organize the sequence of these activities, i.e., 'algorithm ordering mechanism'. None of any existing data cleaning approaches reviewed in chapter 2 has addressed this problem specifically. The order proposed by the 'algorithm ordering mechanism' addresses both effectiveness and efficiency during the data cleaning process.

Finally, the proposed framework supplies a function of 'algorithm selection mechanism' which can provide an optimized algorithm regarding the different factors involved such as problem domain, error rate, computational cost. Compared with existing approaches such as IntelliClean which offer only a fixed solution to cope with all situations or some approach that require its users to make a choice out of multiple algorithms, this is an improvement. For example, Febrl supports a variety of techniques to deal with duplicate record detection. Choosing a suitable technique and setting the corresponding parameters for the selected technique all depend on its user's preference. Febrl does not supply any recommendations or

guidance during the selection. For users who do not have enough knowledge about these techniques, this is a hard job. The proposed ‘algorithm selection mechanism’ aims to fill this gap by supplying an optimized algorithm to deal with different problems with various factors involved. In this way, both effectiveness and automation degree are improved.

- An evaluation of approximate string matching algorithms

Approximate string matching is an important part in many data cleaning approaches which has been well studied for many years, and a variety of approximate string matching techniques have been proposed for string data for the purpose of matching tuples. There is a growing awareness that the high quality of string matching is a key to a variety of applications, such as data integration, text and web mining, information retrieval and search engines. In such applications, matching names is one of the popular tasks. There are a number of name matching techniques available. Unfortunately, there is no existing name matching technique that performs best in all situations. Different techniques perform differently in different situations. An estimate of similarity between strings can vary significantly depending on the domain and specific field under consideration, traditional similarity measures may fail to estimate string similarity correctly. In the past decade, this problem has been challenged by several researchers. However, none of them have undertaken such a comprehensive analysis and comparison that considers the effect on the performance of accuracy and timing caused by the following factors: error rates, type of strings, type of typos, and the size of datasets.

In this thesis, a comprehensive comparison of the five techniques has been carried out based on a series of experiments on different last name and first name datasets. The experimental results confirmed the statement that there is no clear best technique. The size of dataset, the error rate in a dataset, the types of strings in a

dataset and the types of typos in a string will all affect the performance of a selected algorithm.

Regarding the threshold value selection, according to the experimental results, the higher the error rate in the dataset, the lower the threshold value required in order to achieve the best performance. Timing performance based on the five algorithms on different datasets has also been analyzed and compared. Overall, Jaro-Winkler and Jaro are significantly faster than others.

With the help of the experimental results in chapter 5, recommendations on the selection of a suitable algorithm for a particular name matching task are proposed. Compared with the existing recommendations addressed in the previous research, a group of much more detailed recommendations with the corresponding parameters supplied for the recommended algorithms for a practical use and provide useful help in the development of the 'algorithm selection mechanism' in the proposed data cleaning framework are provided.

6.2 Future work

Based on the discussions in former sections, two possible extensions of the current research work are outlined in this section.

The first extension will be focused on the effective database design regarding the data input, for example, the design of data entry interfaces in database applications. As mentioned in the beginning of this work, the quality of any large real world dataset depends on a number of factors, among which the source of data is often the crucial factors. Dirty data can creep in at every step of the process from initial data acquisition to archival storage. Based on the studies of the different types of dirty data, it is discovered that some of them are introduced during the data entry. For

example, according to table 3.4, dirty data types such as DT.17~DT.20 and DT.26 might all be introduced during the data entry process.

On many occasions, it is common that data entry needs to be done by humans manually, who typically extract information from speech or by inputting the data from written or printed sources. During this process, errors in data can often be mitigated through judicious design of data entry interfaces. Traditionally, the specification and maintenance of database integrity constraints are used to prevent the introduction of the dirty data mentioned above such as data type checks, bounds on numeric values, and referential integrity. The most common reason for this behaviour is the enforcement of integrity constraints on the data (rules that ensure completeness and consistency of data entered into the system). These integrity constraints were invented precisely to keep data as clean as possible.

However, the limitation remains that integrity constraints do not prevent bad data and in some cases, constraint enforcement leads to user frustration. For example, the requirement that a field be non-empty is not sufficient to ensure that a user provides meaningful contents. Therefore, an alternative approach is to provide the data entry user with convenient affordances to understand, override and explain constraint violations, thus discouraging the silent injection of bad data, and encouraging annotation of surprising or incomplete source data [131]. According to Hellerstein, several guiding principles for the design of data entry interfaces are proposed [131]. Based on the theoretical analysis, it is shown that a good data entry interfaces will help with preventing the errors from entering into the database. As stated by the old aphorism that an ounce of prevention is worth a pound of cure. Therefore, it is worthwhile extending the research work on the effective database design with respect to the data input.

The second extension of the future research will be focused on the development of a

comprehensive data cleaning tool for database applications based on the framework proposed in this thesis.

The challenge remains the realization of the two mechanisms (AOM and ASM) introduced in the proposed data cleaning framework. Regarding the algorithm ordering mechanism, theoretical analyses are given with ordering the multiple data cleaning tasks during the data cleaning process. It shows how the order of the cleaning of multiple identified dirty data will vary according to the different selection of an algorithm. However, experimental results are not achieved and will be considered as a part of the future work.

Regarding the algorithm selection mechanism, only five selected approximate string matching algorithms are involved for the experiments in this thesis. These five algorithms are the most popular character-level algorithms frequently referenced in most literature. They can be used for dealing with the matching of personal names parsed into single fields such as last name or first name. Although recommendations and a list of figures/numerical values regarding the selection of threshold values for each of these five algorithms are presented. In order to benefit from using these data such as the threshold value suggested in the experimental results, it is assumed that the error rate of a pre-defined dataset should be known. This is a difficult task in practice as users have no idea about the error rate with respect to the data in advance. It is expected that a reasonable method will be available in the future to help with estimating the error rate of a given dataset.

Besides, although relative comparison of accuracy performance among different token-level algorithms exists in the literature [128], the characteristics of the data addressed are not as many as in the presented experiments. The future work will include having these token-level algorithms tested with similar data characteristics addressed in chapter 5. Apart from the five character-level algorithms, other

character-level algorithms mentioned in the Febrl system should also be considered in the future for further experimental works.

Finally, in this thesis, the dirty data type involved in the experimental work is ‘misspelling’. More dirty data types will be involved in the future work for the testing of both character-level algorithms and token-level algorithms such as abbreviation and word swap. The successful outcome of the future work would certainly improve the development of the algorithm selection mechanism and thus, enhance the performance of data cleaning system in database applications.

REFERENCES

[1] Chu, Y.C., Yang, S.S., & Yang, C.C. (2001). Enhancing data quality through attribute-based metadata and cost evaluation in data warehouse environments. *Journal of the Chinese Institute of Engineers*,*24*(4), 497-507.

[2] Pierce, E.M. (2003). *A Progress Report from the MIT Information Quality Conference,* Retrieved March 1, 2012 from http://www.tdan.com/view-articles/5143/

[3] Wang, R.Y., Storey, V.C., & Firth, C.P. (1995). A framework for analysis of data quality research. *IEEE Transactions on Knowledge and Data Engineering, 7*(4), 623-640.

[4] Wang, R.Y., & Strong, D.M. (1996). Beyond Accuracy: What Data Quality Means to Data Consumers. *Journal of Management Information Systems, 1*(12), 5-34.

[5] English, L. (2000). *Plain English on Data Quality*. Retrieved March 1, 2012 from *http://www.information-management.com/issues/20000901/2642-1.html*.

[6] Kim, W., Choi, B., Hong, E.Y., Kim, S.Y., & Lee, D. (2003). A Taxonomy of Dirty Data. *Data Mining and Knowledge Discovery,* 7, 81-99.

[7] Redman, T. (1998). The Impact of Poor Data Quality on the Typical Enterprise. *Communications of the ACM, 41*, 79-82.

[8] Orr, K. (1998). Data Quality and Systems Theory. *Communications of the ACM*, 41, 66-71.

[9] Lee, M. L., Ling, T.W., & Low, W.L. (2000). IntelliClean: a knowledge-based intelligent data cleaner. *Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Boston, MA, USA 290-294, 20-23.

[10] Fellegi, I.P., & Sunter, A.B. (1969). A Theory for Record Linkage. *Journal of the American Statistical Association*,*64*.

[11] Hipp, J., Güntzer, U., & Grimmer, U. (2001). Data Quality Mining - Making a Virtue of Necessity. *Proceedings of the 6th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery,* Santa Barbara, California, 52-57.

[12] Loshin, D. (2009). *Monitoring Data Quality Performance Using Data Quality Metrics.* Retrieved March 1, 2012 from http://www.it.ojp.gov/documents/Informatica_Whitepaper_Monitoring_DQ_Using_Metrics.pdf

[13] Hernandez, M, & Stolfo, S. (1998). Real-world Data is Dirty: Data Cleansing and The merge/purge Problem. *Data Mining and Knowledge Discovery,* 2, 9-37.

[14] Elmagarmid, A.K., Ipeirotis, P.G., & Verykios, V.S. (2007). Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering, 19*, 1-16.

[15] Cadot, M., & Martino, J. (2003). A data cleaning solution by Perl scripts for the KDD Cup 2003 task 2. *ACM SIGKDD Explorations Newsletter*, *5*, 158-159.

[16] Sung, S. Y., Li, Z., & Peng, S. (2002). A fast filtering scheme for large database cleansing. *Proceedings of Eleventh ACM International Conference on Information and Knowledge Management*, 76-83.

[17] Feekin, A., & Chen, Z. (2000). Duplicate detection using k-way sorting method. *Proceedings of ACM Symposium on Applied Computing*, Como, Italy, 323-327.

[18] Galhardas, H. (2001). Data Cleaning: Model, Language and Algorithms. Ph.D, University of Versailles, Saint-Quentin-En-Yvelines.

[19] Zhao, L., Yuan, S.S., Peng, S., & Wang, L.T. (2002). A new efficient data cleansing method. *Proceedings of 13th International Conference on Database and Expert Systems Applications*, 484-493.

[20] Simoudis, E., Livezey, B., & Kerber, R. (1995). Using Recon for Data Cleaning. *Advances in Knowledge Discovery and Data Mining*.

[21] Maletic, J., & Marcus, A. (2000). Data Cleansing: Beynod Integrity Analysis. *Proceedings of The Conference on Information Quality (IQ2000)*, Massachusetts Institute of Technology, 200-209.

[22] Maletic, J., Marcus, A., & Lin, K.L. (2001). Ordinal Association Rules for Error Identification in Data Sets. *Proceedings of Tenth International Conference on Information and Knowledge Management (CIKM 2001)*.

[23] Fox, C., Levitin, A., & Redman, T. (1994). The Notion of Data and Its Quality Dimensions. *Information Processing and Management, 30*, 9-19.

[24] Levitin, A., & Redman, T. (1995). A Model of the Data (Life) Cycles with Application to Quality," *Information and Software Technology, 35*, 217-223.

[25] Strong, D.M., Wang, R.Y., & Lee, Y.W. (1997). Data Quality in Context *Communications of the ACM,40*(5), 103-110.

[26] Svanks, M.I. (1988). Integrity Analysis: Methods for Automating Data Quality Assurance. *Information and Software Technology,30*(10).

[27] Müller, H., & Freytag, J.C. (2003). Problems, Methods, and Challenges in Comprehensive Data Cleansing. *Tech. Rep*, HUB-1B-164.

[28] Rahm, E., & Do, H. (2000). Data Cleaning: Problems and Current Approaches. *IEEE Bulletin of the Technical Committee on Data Engineering,23*(4), 3-13.

[29] Kim, W. (2002). On three major holes in Data Warehousing Today. *Journal of Object Technology, 1*, 2002.

[30] Oliveira, P., Rodrigues, F.T., Henriques, P., & Galhardas, H. (2005). A Taxonomy of Data Quality Problems. *Second International Workshop on Data and Information Quality (in conjunction with CAISE'05)*, Porto, Portugal.

[31] Luebbers, D., Grimmer, U., & Jarke, M. (2003). Systematic Development of Data Mining-Based Data Quality Tools. *The 29th International Conference on Very Large Databases*, Berlin, Germany.

[32] Peng, T. (2008). A Framework for Data Cleaning in Data Warehouses. *Proceeding of ICEIS 2008*, Spain, 473-478.

[33] Hellerstein, J.M., & Raman, V. (2001). Potter's Wheel: An Interactive Framework for Data Transformation and Cleaning. *Proceedings of the 27th VLDB Conference*, Roma, Italy.

[34] Schallehn, E., & Sattler, K.U. (2001). A Data Preparation Framework based on a Multidatabase Language. *International Database Engineering Applications Symposium (IDEAS)*, Grenoble, France.

[35] Joachims, T. (1999). *Making large-scale svm learning practical*: MIT Press.

[36] Cohen, W.W., & Richman, J. (2002). Learning to match and cluster large high-dimensional data sets for data integration. *Proceeding of Eighth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD '02)*.

[37] McCallum, A., & Wellner, B. (2004). Conditional models of identity uncertainty with application to noun coreference. *The Advances in Neural Information Processing Systems (NIPS'04)*.

[38] Domingos, P. (2004). Multi-relational record linkage. *KDD-2004 Workshop Multi-Relational Data Mining*.

[39] Pasula, H., Milch, B., Marthi, B., Russell, S., & Shpitser, I. (2002). Identity uncertainty and citation matching. *The Advances in Neural Information Processing Systems (NIPS'02)*.

[40] Lim, E., Srivastava, J., Prabhakar, S., & Richardson. (1993). Entity identification in database integration. *Ninth IEEE Int'l Conf. Data Eng. (ICDE '93)*, 294-301.

[41] Galhardas, H., Shasha, D., & Florescu, D. (2001). Declarative data cleaning: Language, model, and algorithms. *27th Int'l Conf. Very Large Databases (VLDB '01)*.

[42] Guha, S., Koudas, N., Marathe, A., & Srivastava, D. (2004). Merging the results of approximate match operations, *30th Int'l Conf. Very Large Databases (VLDB '04)*.

[43] Ananthakrishna, R., Chaudhuri, S., & V.Ganti. (2002). Eliminating fuzzy duplicates in data warehouses. *28th Int'l Conf. Very Large Databases (VLDB '02)*.

[44] Chaudhuri, S., Ganti, V., & Motwani, R. (2005). Robust identification of fuzzy duplicates. *21st IEEE Int'l Conf. Data Eng. (ICDE '05)*.

[45] Vassiliadis, P., Vagena, Z., Skiadopoulos, S., Karayannidis, N., & T. Sellis. (2001). ARKTOS: towards the modeling, design, control and execution of ETL Processes. *Information Systems,26*, 537-561.

[46] Christen, P. (2008). Febrl:an open source data cleaning, deduplication and record linkage system with a graphical user interface. *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*.

[47] Christen, P. (2009). Development and user experiences of an open source data cleaning, deduplication and record linkage system, *SIGKDD Explorations Newsletter, 11*(1), 2009.

[48] English, L. (1999). *Improving Data Warehouse and Business Information Quality: methods for reducing costs and increasing profits,* John Wiley & Sons, Inc.

[49] Parssian, A., Sarkar, S., & Jacob, V.S. (1999). Assessing Data Quality For Information Products. *Proceeding ICIS '99 Proceedings of the 20th international conference on Information Systems*.

[50] Pipino, L.L., Lee, Y.W., & Wang, R.W. (2002). Data Quality Assessment. *Communications of the ACM,45*, 211-218.

[51] Madnick, S.E., Wang, R.Y., Lee, Y.W. & Zhu, H. (2009). Overview and Framework for Data and Information QUality Research. *Journal of Data and Information Quality (JDIQ), 1*(1).

[52] Blake, R. & Mangiameli, P. (2011). The Effects and Interactions of Data Quality and Problem Complexity on Classification. *Journal of Data and Information Quality, 2*(2), 2011.

[53] Tayi, G., & Ballou, D.P. (1998). Examining Data Quality, *Communications of the ACM, 41*, 54-57.

[54] Kahn, B.K., Strong, D.M., & Wang, R.W. (2002). Information Quality Benchmark: Product and service performance. *Communications of the ACM, 45*, 184-192.

[55] Drucker, P.F. (1985). Playing in the Information-Based 'Orchestra'. *The Wall Street Journal*.

[56] Juran, J.M. (1988). *Juran on planning for quality*. New York: The Free Press.

[57] Juran, J.M. (1999). How to think about Quality. In J.M.Juran & A.B.Godfrey (Eds) *Juran's quality handbook* (pp.2.1-2.18), New York: McGraw-Hill.

[58] Ballou, D.P., & Pazer, H. (1985). Modeling data and process quality in multi-input, multi-output information systems, *Manage. Sci. 31*,150-162.

[59] Wang, R., & Firth, C.P. (1996). *Data Quality Systems: Evaluation and Implementation*. London: Cambridge Market Intelligence Ltd.

[60] Kriebel, C.H. (1979). Evaluating the quality of information systems. In N.Szysperski and E.Grochla (eds.), *Design and Implementation of Computer Based Information Systems*. Germantown, PA: Sijthtoff and Noordhoff.

[61] Wang, R, Strong, D, & Guarascio, L. (1994). An empirical investigation of data quality dimensions: A data consumer's perspective. MIT TDQM Research Program E53-320 TDQM-94-01.

[62] Wand, Y., & Wang, R.Y. (1996). Anchoring data quality dimensions in ontological foundations. *Communication of the ACM, 39*, 86-95.

[63] Redman, T.C. (1997). *Data Quality for the Information Age*: Artech House.

[64] Bovee, M., Srivastava, R.P., & Mak, B. (2001). A Conceptual Framework and Belief-Function Approach to Assessing Overall Information Quality. *Proceedings of the Sixth International Conference on Information Quality*, Boston, MA.

[65] Naumann, F. (2002). *Quality-Driven Query Answering for Integrated Information Systems*. Springer-Verlag, Berlinm Heidelberg.

[66] Wang, K.Q., Tong, S.R., Roucoules, L., & Eynard, B. (2008). Analysis of Data Quality and Information Quality Problems in Digital Manufacturing. *Proceedings of the 2008 IEEE ICMIT*.

[67] Jarke, M., Jeusfeld, M., Quix, C., & Vassiliadis, P. (1999). Architecture and Quality in Data Warehouses: an Extended repository Approach. *Information Systems, 24*.

[68] Scannapieco, M., & Catarci, T. (2002). Data Quality under a Computer Science Perspective. *Italian: Archivi & Computer*.

[69] Liu, L. (2002). Evolutionary Data Quality. *7th International Conference on Information Quality (IQ 2002)*.

[70] Jarke, M., Lenzerini, M., Vassiliou, Y., & Vassiliadis, P. (2001). *Fundamentals of Data Warehouses. 2nd Edition*. Springer-Verlag Berlin and Heidelberg GmbH & Co.K.

[71] Lee, Y.W., Strong, D.M., Kahn, B.K., & Wang, R.Y. (2002). AIMQ: A Methodology for Information Quality Assessment. *Information and Management,40*, 133-46.

[72] Zmud, R. (1978). Concepts, Theories and Techniques: An Empirical

Investigation of the Dimensionality of the Concept of Information. *Decision Sciences, 9*, 187-195.

[73] Delone, W., & McLean, E. (1992). Information Systems Success: The Quest for the Dependent Variable. *Information Systems Research,3*, 60-95.

[74] Goodhue, D.L., (1995). Understanding User Evaluations of Information Systems Management Science. *INFORMS,41*(12), 1827-1844.

[75] Jarke, M., & Vassiliou, Y. (1997). Data Warehouse Quality: A Review of the DWQ Project. *Proceedings of the 1997 Conference on Information Quality*, Cambridge, MA, 299-313.

[76] Matsumura, A., & Shouraboura, N. (1996). Competing with Quality Information. *Proceedings of the 1996 Conference on Information Quality*, Cambridge, MA, 72-86.

[77] Gardyn, E. (1997). A Data Quality Handbook for a Data Warehouse. *Proceedings of the 1997 Conference on Information Quality*, Cambridge, MA, 267-290.

[78] Redman, T.C. (1992). *Data Quality: Management and Technology*. New York, NY: Bantam Books.

[79] Kovac, R., Lee., Y & Pipino, L (1997). Total Data Quality Management: The Case of IRI. *Proceedings of the 1997 Conference on Information Quality*, Cambridge, MA, 63-79.

[80] Meyen, D & Willshire, M. (1997). A Data Quality Engineering Framework. *Proceedings of the 1997 Conference on Information Quality*, Cambridge, MA, 95-116.

[81] Tsichritzis, D & Lochovsky, F. (1982). *Data models*: Englewood Cliffs, NJ:Prentice-Hall.

[82] Melissadata (2012). *Why Dirty Data May Cost You $180,000*. Retrieved March 1, 2012 from http://www.melissadata.com/enews/articles/1206/1.htm.

[83] Huang, K., Lee, Y.W. & Wang, R.Y (1999). *Quality information and knowledge,*: Prentice Hall.

[84] Missier, P., Lalk, G., Verykios, V., Grillo, F., Lorusso, T., & Angeletti, P. (2003). Improving Data Quality in Practice: A Case Study in the Italian Public Administration. *Journal of Distributed and Parallel Databases, 13*(2).

[85] Wang, R.Y., Lee, Y.W. & Ziad, M. (2001). *Data Quality*. Heidelberg Springer.

[86] Fisher, C.W., Kingma, B.R. (2001). Criticality of Data Quality as Exemplified in Two Disasters. *Information & Management,39*, 109-116.

[87] Eppler, M., & Helfert, M. (2004). A Classification and Analysis of Data Quality Costs. *The Ninth International Conference on Information Quality*, MIT.

[88] C. A. I. Board, (2003). "Columbia Accident Investigation Board report," Washington, DC: U.S.

[89] Adelman, L. M. S., & Abai, M. (2005). *Data Strategy*: Addison-Wesley Professional.

[90] Cappiello, C., & Francalanci, C. (2002). *DL4B Considerations about Costs Deriving from a Poor Data Quality*, Retrieved March 1, 2012 from http://www.dis.uniroma1.it/~dq/docs/Deliverables/DL4B.pdf.

[91] Eppler, M. (2003). *Managing Information Quality*. Springer.

[92] Kahn, B., Katz-Haas, R., & Strong, D.M. (2000). How to get an Information Quality Program Started: The Ingenix Approach. *Proceedings of the 2000 Conference on Information Quality*, 28-35.

[93] Naumann, F., & Rolker, C. (2000). *Assessment Methods for Information Quality Criteria*.

[94] Segev, A & Wang, R (2001). Data Quality Challenges in Enabling eBusiness Transformation. *Proceedings of the Sixth International Conference on Information Quality*, 83-91.

[95] Strong, D.M., Lee, Y.W., & Wang, R.Y (1997). 10 Potholes in the Road to Information Quality. *Computer IEEE, 30*, 38-46.

[96] Verykios, V.S., Elfeky, M.G., Elmagarmid, A.K., Cochinwala, M., & Dalal, S. (2000). On the Accuracy and Completeness of the Record Matching Process. *Proceedings of the 2000 Conference on Information Quality*.

[97] Kim, W., & Choi, B (2003). Towards Quantifying Data Quality Costs. *Journal of Object Technology*, 2 , 69-76.

[98] Ballou, D., & Pazer, H. (1995). Designing information systems to optimize the accuracy-timeliness trade off. *Information Systems Research*, *6*, 51-72.

[99] Ballou, D.P., & Tayi, G.K. (1989). Methodology for allocating resources for data quality enhancement, *Communications of the ACM*, *32*, 320-329.

[100] Ballou, D., Wang, R., Pazer, H., & Tayi, G. (1998). Modeling information manufacturing systems to determine information product quality. *Management Science*, *44*, 462-484.

[101] Batini, C., Cappiello, C., Francalanci, C., & Maurino, A. (2009). Methodologies for data quality assessment and improvement. *ACM Computing SUrveys (CSUR)*.

[102] Even, A., & Shankaranarayanan, G. (2009). Dual Assessment of Data Quality in Customer Databases, *Journal of Data and Information Quality*, *1*(3).

[103] Strong, D.M. (1997). IT process designs for improving information quality and reducing exception handling: a simulation experiment. *Information and Management, 31*, 251-263.

[104] Strong, D.M. (1997). Total Data Quality Management Program, *Proceedings of the Conference on Information Quality*, Cambridge, MA, 372.

[105] Brown, S.M. (1997). Preparing Data for the Data Warehouse.," in *Proceedings of the 1997 Conference on Information Quality*, Cambridge, MA, 1997, pp. 291-298.

[106] Schusell, G. (1997). Data quality the top problem, DW for Data Warehousing Management, Digital Consulting Institute.

[107] Deming, E.W. (1986). *Out of the Crisis*. Cambridge, Mass: MIT Center for Advanced Engineering Study.

[108] Ge, M., & Helfert, M. (2008). Data and Information Quality Assessment in Information Manufacturing Systems. *The 11th International Conference on Business Information Systems (BIS2008)*, 380-389.

[109] Cappiello, C., & Francalanci, C., & Pernici, B. (2003). Time-Related Factors of Data Quality in Multichannel Information Systems *Journal of Management Information Systems, 20*,71-91.

[110] Cooper, R.B. (1983). Decision production: A step toward a theory of

managerial information requirements. *Proceeding Fourth Int'l Conf on Information Systems*, Houston, 215-268.

[111] Emery, J.C. (1969). *Organizational planning and control systems: Theory and technology*: New York: Macmillan.

[112] Francalanci, C. & Pernici, B. (2004). Data quality assessment from the user's perspective. *Proceedings of the 2004 international workshop on Information quality in information systems*.

[113] Lee, Y.W., Pipino, L.L., Funk, J.D., & Wang, R.W. (2009). *Journey to Data Quality*, The MIT Press.

[114] Chanana, V., & Koronios, A. (2007). Data Quality through Business Rules, *International Conference on Information & Communication Technology ICICT 2007*, 262-265.

[115] Morgan, T. (2002). *Business Rules and Information Systems: Aligning IT with Business Goals*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA.

[116] Halle, B.V. & Ross, R.G. (2001). *Business Rule Applied: Building Better Systems Using Business Rules Approach*, John Wiley & Sons, Inc. New York, NY, USA.

[117] Kim, W., & Seo, J. (1991). On classifying schematic and data heterogenety in multidatabase systems. *IEEE Computer,24*(12), 12-18.

[118] Hermansen, J. (1985). Automatic Name Searching in Large Databases of International Names, Georgetown University Dissertation, Washington, DC.

[119] Smith, T.F., & Waterman, M.S. (1981). Identification of Common Molecular Subsequences, *J. Mol. Biol*, *147*, 195-197.

[120] Jaro, M. (1989). Advances in Record-Linkage Methodology as Applied to Matching the 1985 Census of Tampa,Florida. *Journal of the American Statistical Associations*, *89*, 414-420.

[121] Navarro, G. (2001). A guided tour to approximate string matching, *ACM Computing Surveys*, 33, 31-88.

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

[123] Winkler, W. (1990). String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage. *Proceedings of the Section on Survey Research Methods*, 354-359.

[124] Christen, P. (2006). A Comparison of Personal Name Matching: Techniques and Practical Issues," in *Proceedings of the Sixth IEEE International Conference on Data Mining - Workshops (ICDMW '06)*, Washington, DC, USA, 2006, pp. 290-294.

[125] Li, L., Peng, T., & Kennedy, J. (2011). A Rule Based Taxonomy of Dirty Data, *GSTF International Journal on Computing*, *1*.

[126] Bilenko, M., Mooney, R., Cohen, W., Ravikumar, P., & Fienberg, S. (2003) Adaptive Name Matching in Information Integration. *IEEE Intelligent Systems*,*18*, 16-23.

[127] Cohen, W., & Fienberg, S. (2003) A comparison of string distance metrics for name-matching tasks. *Proceedings of the IJCAI-2003 Workshop on Information Integration on the Web*, 73-78.

[128] Hassanzadeh, O., Sadoghi, M., & Miller, R. (2007). Accuracy of Approximate String Joins Using Grams. *Proceedings of QDB'2007*, 11-18.

[129] Patman, F., & Thompson, P. (2003). Names: A New Frontier in text mining, *ISI-2003, Springer LNC 2665,* 27-38.

[130] Rijsbergen, C. (1979) *Information Retrieval*, 2nd ed., London Butterworths.

[131] Hellerstein, J. M. (2008). Quantitative data cleaning for large databases. Report for United Nations Economic Commission for Europe: EECS Computer Science Division.

APPENDIX A

A.1: Data of the maximum F score for different techniques on the different last name datasets

Algorithm	F score (Low ER)	F score (Medium ER)	F score (High ER)	Dataset
Levenshtein	0.8872	0.7209	0.6808	9454
Jaro	0.8872	0.7401	0.6773	9454
Jaro-Winkler	0.8848	0.7275	0.7053	9454
Q-gram	0.8872	0.6622	0.5813	9454
Smith-Waterman	0.2164	0.2095	0.2032	9454
Levenshtein	0.8813	0.7293	0.679	7154
Jaro	0.8813	0.7541	0.7242	7154
Jaro-Winkler	0.8806	0.7666	0.7375	7154
Q-gram	0.8813	0.6735	0.6322	7154
Smith-Waterman	0.3166	0.3078	0.2999	7154
Levenshtein	0.8868	0.7258	0.6836	5000
Jaro	0.8868	0.7461	0.7278	5000
Jaro-Winkler	0.8854	0.7548	0.7394	5000
Q-gram	0.8868	0.6644	0.6122	5000
Smith-Waterman	0.2951	0.2813	0.28	5000
Levenshtein	0.8895	0.715	0.6516	3600
Jaro	0.8895	0.7349	0.7163	3600
Jaro-Winkler	0.8878	0.7337	0.706	3600
Q-gram	0.8895	0.665	0.5757	3600

Smith-Waterman	0.2949	0.2822	0.2718	3600
Levenshtein	0.8862	0.7265	0.6722	2300
Jaro	0.8862	0.7383	0.7012	2300
Jaro-Winkler	0.8852	0.7252	0.706	2300
Q-gram	0.8862	0.6626	0.5425	2300
Smith-Waterman	0.3118	0.2833	0.2802	2300
Levenshtein	0.887	0.7072	0.6328	1000
Jaro	0.887	0.7161	0.6315	1000
Jaro-Winkler	0.8877	0.6857	0.6344	1000
Q-gram	0.887	0.6609	0.5181	1000
Smith-Waterman	0.312	0.258	0.2359	1000
Levenshtein	0.8874	0.7254	0.6768	500
Jaro	0.8862	0.7199	0.7022	500
Jaro-Winkler	0.8874	0.7411	0.682	500
Q-gram	0.8862	0.6623	0.5617	500
Smith-Waterman	0.3748	0.3443	0.3266	500
Levenshtein	0.892	0.7833	0.8089	200
Jaro	0.892	0.7482	0.7368	200
Jaro-Winkler	0.892	0.7325	0.7045	200
Q-gram	0.892	0.7653	0.7406	200
Smith-Waterman	0.5949	0.5177	0.5077	200

Table A.1 Accuracy results for last name datasets

A.2: Data of threshold value selection for each technique to obtain the maximum F score in different last name datasets

Algorithm	Low Error Rate	Medium Error Rate	High Error Rate	Data Size
Levenshtein	0.99	0.85	0.8	9454
Levenshtein	0.99	0.85	0.8	7154
Levenshtein	0.99	0.85	0.8	5000
Levenshtein	0.99	0.85	0.8	3600
Levenshtein	0.99	0.85	0.8	2300
Levenshtein	0.99	0.85	0.8	1000
Levenshtein	0.9	0.85	0.8	500
Levenshtein	0.99	0.8	0.75	200
Jaro	0.99	0.95	0.9	9454
Jaro	0.99	0.95	0.9	7154
Jaro	0.99	0.95	0.9	5000
Jaro	0.99	0.95	0.9	3600
Jaro	0.99	0.95	0.9	2300
Jaro	0.99	0.95	0.9	1000
Jaro	0.99	0.9	0.9	500
Jaro	0.99	0.9	0.9	200
Jaro-Winkler	0.99	0.95	0.95	9454
Jaro-Winkler	0.99	0.95	0.95	7154
Jaro-Winkler	0.99	0.95	0.95	5000
Jaro-Winkler	0.99	0.95	0.95	3600
Jaro-Winkler	0.99	0.95	0.95	2300
Jaro-Winkler	0.99	0.95	0.95	1000
Jaro-Winkler	0.99	0.95	0.95	500
Jaro-Winkler	0.99	0.95	0.9	200
Q-gram	0.99	0.99	0.8	9454
Q-gram	0.99	0.99	0.8	7154
Q-gram	0.99	0.99	0.8	5000
Q-gram	0.99	0.99	0.8	3600
Q-gram	0.99	0.99	0.8	2300
Q-gram	0.99	0.99	0.8	1000
Q-gram	0.99	0.99	0.8	500
Q-gram	0.99	0.8	0.75	200

Smith-Waterman	0.9	0.9	0.85	9454
Smith-Waterman	0.99	0.9	0.9	7154
Smith-Waterman	0.9	0.9	0.9	5000
Smith-Waterman	0.9	0.9	0.85	3600
Smith-Waterman	0.9	0.9	0.9	2300
Smith-Waterman	0.9	0.85	0.85	1000
Smith-Waterman	0.9	0.85	0.85	500
Smith-Waterman	0.99	0.85	0.85	200

Table A.2 Threshold value selection for last name datasets

A.3: Data of average time cost for the five techniques on four different sizes of datasets (9454, 7154, 5000, and 3600)

Algorithm	Time cost Low ER	Time cost Medium ER	Time cost High ER	Dataset
Levenshtein	128.945	130.82	143.751	9454
Jaro	57.573	57.977	61.123	9454
Jaro-Winkler	42.299	42.418	46.998	9454
Q-gram	162.773	166.503	174.301	9454
Smith-Waterman	220.535	225.116	227.98	9454
Levenshtein	78.745	79.665	82.609	7154
Jaro	35.264	34.584	37.648	7154
Jaro-Winkler	26.829	26.062	28.315	7154
Q-gram	98.102	98.906	101.235	7154
Smith-Waterman	147.848	134.764	145.340	7154
Levenshtein	37.023	41.007	40.255	5000
Jaro	16.514	17.664	17.736	5000
Jaro-Winkler	12.612	13.031	13.166	5000
Q-gram	47.13	48.85	50.507	5000
Smith-Waterman	65.767	69.181	70.237	5000
Levenshtein	19.9782	21.7713	22.63736	3600
Jaro	9.160714	9.777714	10.067	3600
Jaro-Winkler	6.887571	7.565857	7.578571	3600
Q-gram	24.538	26.62	27.89082	3600
Smith-Waterman	34.06429	39.15488	38.58414	3600

Table A.3 Time cost in last name datasets

A.4: Data of the maximum F score for the 2300 first name datasets with the three different types of typos

Algorithm	F score (Low ER)	F score (Medium ER)	F score (High ER)	Type of typo
Levenshtein	0.8999	0.8177	0.8394	TFP
Jaro	0.8912	0.7868	0.7936	TFP
Jaro-Winkler	0.8865	0.7763	0.7705	TFP
Q-gram	0.8887	0.6667	0.6978	TFP
Smith-Waterman	0.3661	0.3437	0.359	TFP
Levenshtein	0.8961	0.7884	0.7902	TLP
Jaro	0.9032	0.8195	0.7848	TLP
Jaro-Winkler	0.9062	0.8148	0.82	TLP
Q-gram	0.8863	0.7662	0.7861	TLP
Smith-Waterman	0.3552	0.3305	0.3371	TLP
Levenshtein	0.8909	0.7259	0.6961	TR
Jaro	0.8867	0.7708	0.7947	TR
Jaro-Winkler	0.8846	0.7844	0.7722	TR
Q-gram	0.8867	0.6645	0.5849	TR
Smith-Waterman	0.3434	0.3327	0.3414	TR

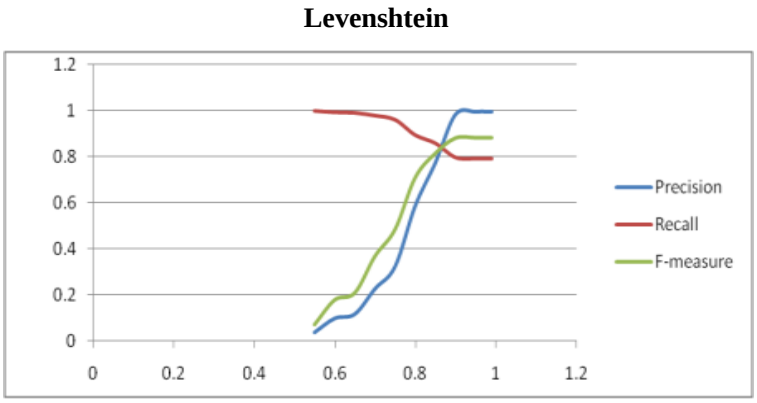
Table A.4 Accuracy results for 2300 first name datasets with different typos

A.5: Data of the maximum F score for the 2300 last name datasets with the three different types of typos

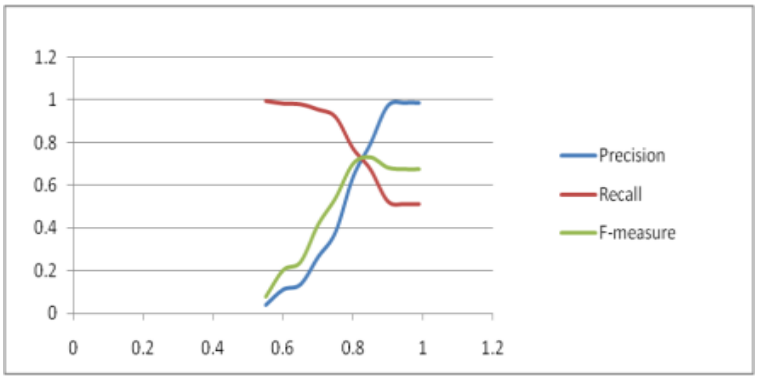
Algorithm	F score (Low ER)	F score (Medium ER)	F score (High ER)	Type of typo
Levenshtein	0.89	0.7346	0.734	TFP
Jaro	0.8889	0.7524	0.6918	TFP
Jaro-Winkler	0.8876	0.7124	0.7057	TFP
Q-gram	0.8889	0.6663	0.5807	TFP
Smith-Waterman	0.3533	0.3225	0.3317	TFP
Levenshtein	0.8896	0.7365	0.7422	TLP
Jaro	0.8885	0.7674	0.7405	TLP
Jaro-Winkler	0.8989	0.7409	0.7668	TLP
Q-gram	0.8885	0.6663	0.7162	TLP
Smith-Waterman	0.3537	0.3633	0.39	TLP
Levenshtein	0.8862	0.7265	0.6722	TR
Jaro	0.8862	0.7383	0.7012	TR
Jaro-Winkler	0.8852	0.7252	0.706	TR
Q-gram	0.8862	0.6626	0.5425	TR
Smith-Waterman	0.3118	0.2833	0.2802	TR

Table A.5 Accuracy results for 2300 last name datasets with different typos

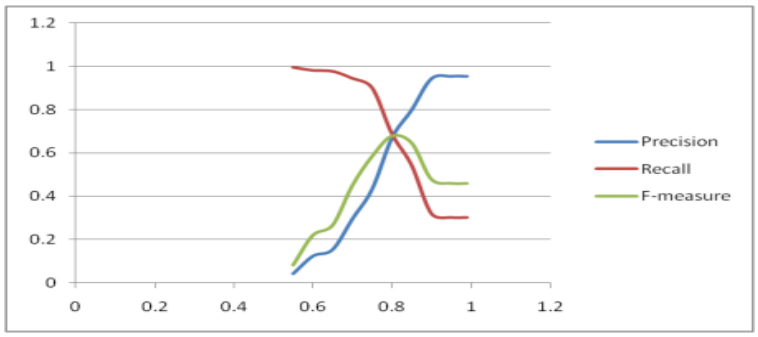
A.6: Accuracy relative to the value of threshold on different last name datasets with different error rates for levenshtein algorithm



(a) Low Error Dataset



(b) Medium Error Dataset

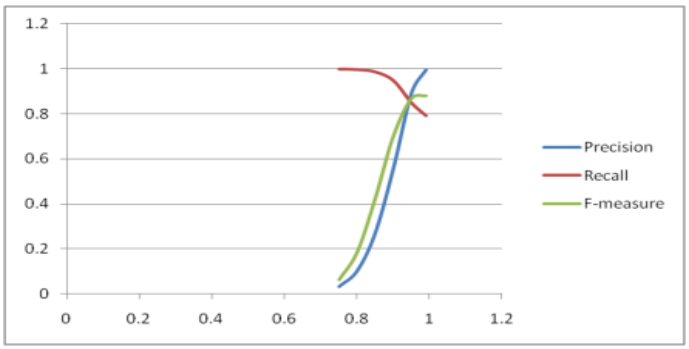


(c) High Error Dataset

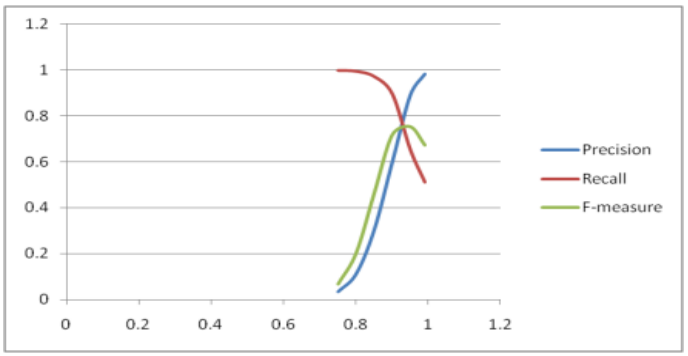
Fig.A.1 Accuracy relative to threshold value for Levenshtein algorithm

A.7: Accuracy relative to the value of threshold on different last name datasets with different error rates for Jaro algorithm

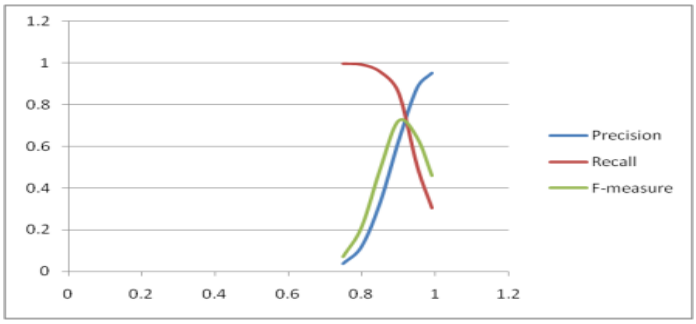
Jaro



(a) Low Error Dataset



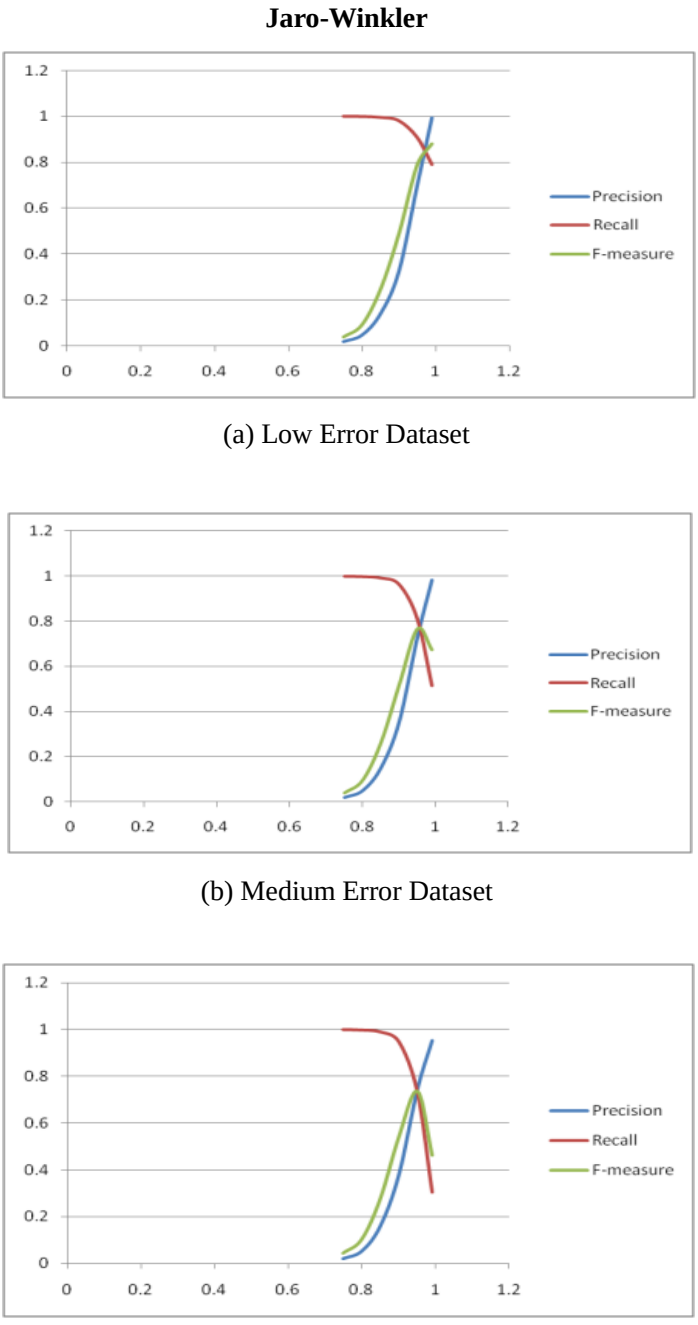
(b) Medium Error Dataset



(c) High Error Dataset

Fig.A.2 Accuracy relative to threshold value for Jaro algorithm

A.8: Accuracy relative to the value of threshold on different last name datasets with different error rates for Jaro-Winkler algorithm



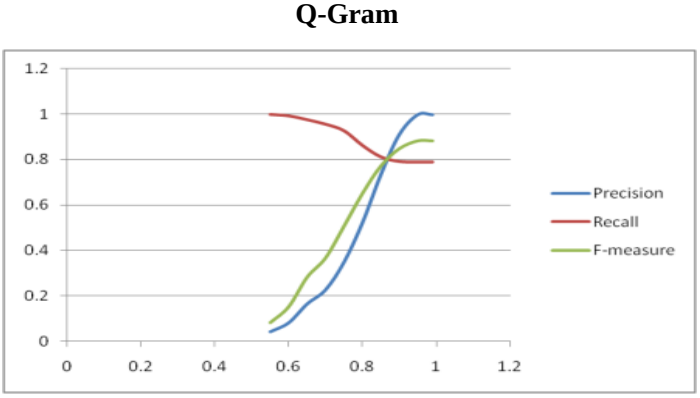
(a) Low Error Dataset

(b) Medium Error Dataset

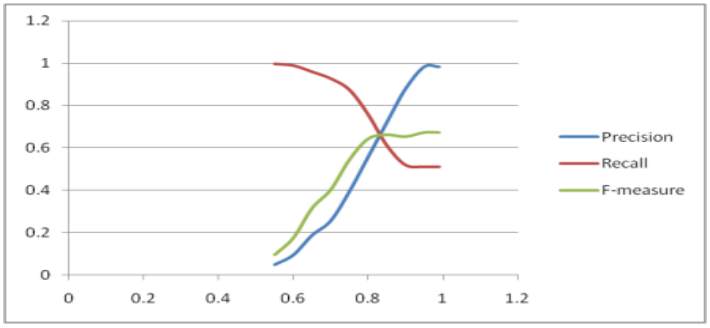
(c) High Error Dataset

Fig.A.3 Accuracy relative to threshold value for Jaro-Winkler algorithm

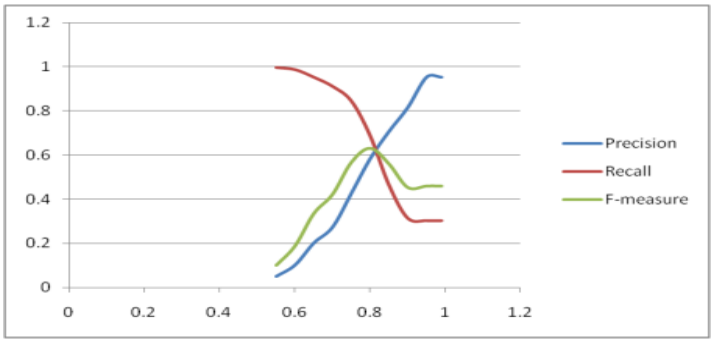
A.9: Accuracy relative to the value of threshold on different last name datasets with different error rates for Q-Gram algorithm



(a) Low Error Dataset



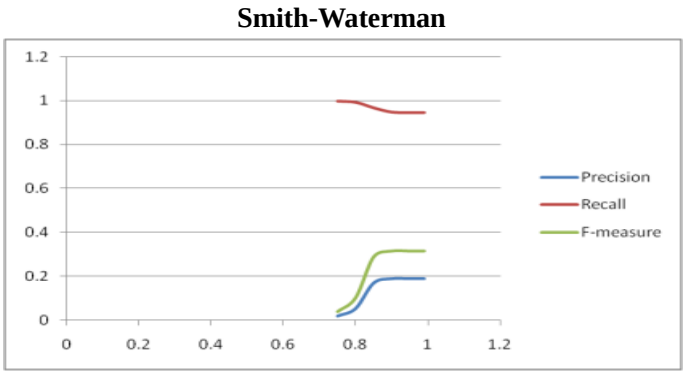
(b) Medium Error Dataset



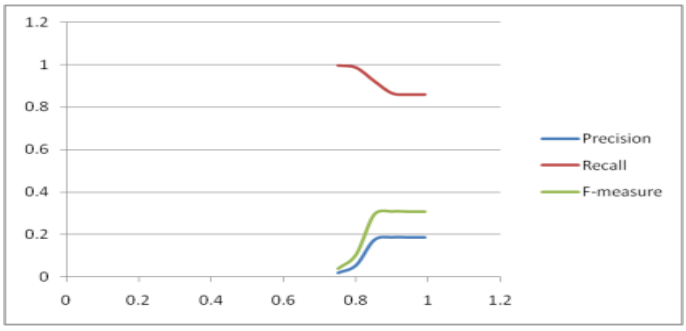
(c) High Error Dataset

Fig.A.4 Accuracy relative to threshold value for Q-Gram algorithm

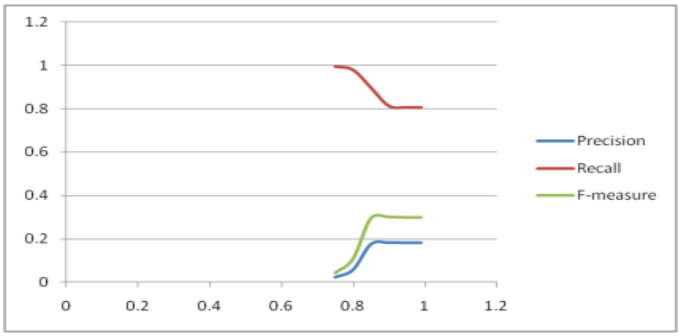
A.10: Accuracy relative to the value of threshold on different last name datasets with different error rates for Smith-Waterman algorithm



(a) Low Error Dataset



(b) Medium Error Dataset



(c) High Error Dataset

Fig.A.5 Accuracy relative to threshold value for Smith-Waterman algorithm

A.11: Algorithm selection for last name and first name datasets

Size of dataset	Error rate	Algorithm	Threshold value	Type of string
9454	Low	Jaro	0.99	Last name
9454	Medium	Jaro	0.95	Last name
9454	High	Jaro-Winkler	0.95	Last name
7154	Low	Jaro	0.99	Last name
7154	Medium	Jaro-Winkler	0.95	Last name
7154	High	Jaro-Winkler	0.95	Last name
5000	Low	Jaro	0.99	Last name
5000	Medium	Jaro-Winkler	0.95	Last name
5000	High	Jaro-Winkler	0.95	Last name
3600	Low	Jaro	0.99	Last name
3600	Medium	Jaro	0.95	Last name
3600	High	Jaro	0.9	Last name
2300	Low	Jaro	0.9	Last name
2300	Medium	Jaro	0.95	Last name
2300	High	Jaro-Winkler	0.95	Last name
1000	Low	Jaro-Winkler	0.99	Last name
1000	Medium	Jaro	0.95	Last name
1000	High	Jaro-Winkler	0.95	Last name
500	Low	Jaro-Winkler	0.99	Last name
500	Medium	Jaro-Winkler	0.95	Last name
500	High	Jaro	0.9	Last name
200	Low	Jaro-Winkler	0.99	Last name
200	Medium	Levenshtein	0.8	Last name
200	High	Levenshtein	0.75	Last name
2300	Low	Jaro-Winkler	0.99	First name (TLP)

2300	Medium	Jaro	0.95	First name (TLP)
2300	High	Jaro-Winkler	0.95	First name (TLP)
2300	Low	Levenshtein	0.9	First name (TFP)
2300	Medium	Levenshtein	0.8	First name (TFP)
2300	High	Levenshtein	0.8	First name (TFP)
2300	Low	Levenshtein	0.9	First name (TR)
2300	Medium	Jaro-Winkler	0.95	First name (TR)
2300	High	Jaro	0.9	First name (TR)

Table A.6 Algorithm selection and Threshold values for last name datasets and first name datasets

APPENDIX B

B.1: Business entity rules

Business Entity Rules	Sub rules
R1.1 Entity uniqueness rules	R1.1.1 Primary key rule: every instance of a business entity has its own unique identifier.
	R1.1.2 Primary key can never be NULL.
	R1.1.3 A composite key must be minimal
	R1.1.4 A composite primary key can contain one or more foreign keys
R1.2 Entity cardinality rules	R1.2.1 One-to-one cardinality rule
	R1.2.2 One-to-many (or many-to-one) cardinality rule
	R1.2.3 Many-to-many cardinality rule
R1.3 Entity optionality rules	R1.3.1 One-to-one optionality rule
	R1.3.2 One-to-zero (or zero-to-one) optionality rule
	R1.3.3 Zero-to-zero optionality rule
	R1.3.4 Every instance of an entity that is being referenced by another entity in the relationship must exist.
	R1.3.5 The reference attribute does not have to be known when an optional relationship is not instantiated, i.e., the foreign key can be NULL on an optional relationship.

Table B.1 Business entity rules

B.2: Business attribute rules

Business attribute rules	Sub rules
R2.1 Data inheritance rules	R2.1.1 All generalized business attributes of the supertype are inherited by all subtypes.
	R2.1.2 The unique identifier of the supertype is the same unique identifier of its subtypes.
	R2.1.3 All business attributes of a subtype must be unique to that subtype only.
R2.2 Data domains rules	R2.2.1 Data values should belong to the given list of values.
	R2.2.2 Data values should be within the given range of values.
	R2.2.3 Data values should conforms to the given constrains.
	R2.2.4 Data values contains only a set of allowable characters.
	R2.2.5 Data values should follows the given patterns.

Table B.2 Business attribute rules

B.3: Data dependency rules

Data dependency rules	Sub rules
R3.1 Entity-relationship rules	R3.1.1 The existence of a data relationship depends on the state (condition) of another entity that participates in the relationship.
	R3.1.2 The existence of one data relationship mandates that another data relationship also exists.
	R3.1.3 The existence of one data relationship prohibits the existence of another data relationship.
R3.2 Attribute dependency rules	R3.2.1 The value of one business attribute depends on the state (condition) of the entity in which the attributes exist.
	R3.2.2 The correct value of one attribute depends on, or is derived from, the values of two or more other attributes.
	R3.2.3 The allowable value of one attribute is constrained by the value of one or more other attributes in the same business entity or in a different but related business entity.
	R3.2.4 The existence of one attribute value prohibits the existence of another attribute value in the same business entity or in a different but related business entity.

Table B.3 Data dependency rules

B.4: Data validity rules

Data validity rules	Sub rules
R4.1 Data completeness rules	R4.1.1 All instances exist for all business entities, i.e., all records or rows are present.
	R4.1.2 Referential integrity exists among all referenced business entities.
	R4.1.3 All business attributes for each business entity exist, i.e., all columns are present.
	R4.1.4 All business attributes contain allowable values including NULL when it is allowed.
R4.2 Data correctness rules	R4.2.1 All data values for a business attribute must be correct and representative of the attribute's definition.
	R4.2.2 All data values for a business attribute must be correct and representative of the attribute's specific individual domains.
	R4.2.3 All data values for a business attribute must be correct and representative of the attribute's applicable business rules.
	R4.2.4 All data values for a business attribute must be correct and representative of the attribute's supertype inheritance.
	R4.2.5 All data values for a business attribute must be correct and representative of the attribute's identity rule.
R4.3 Data accuracy rules	R4.3.1 All data values for a business attribute must be accurate in terms of the attribute's dependency rules.
	R4.3.2 All data values for a business attribute must be accurate in terms of the attribute's state in the real world.

R4.4 Data precision rules	R4.4.1 All data values for a business attribute must be as precise as required by the attribute's business requirements.
	R4.4.2 All data values for a business attribute must be as precise as required by the attribute's business rules.
	R4.4.3 All data values for a business attribute must be as precise as required by the attribute's intended meaning.
	R4.4.4 All data values for a business attribute must be as precise as required by the attribute's intended usage.
	R4.4.5 All data values for a business attribute must be as precise as required by the attribute's precision in the real world.
R4.5 Data uniqueness rules	R4.5.1 Every business entity instance must be unique.
	R4.5.2 Every business entity must have only one unique identifier.
	R4.5.3 Every business attribute must have only one unique definition.
	R4.5.4 Every business attribute must have only one unique name.
	R4.5.5 Every business attribute must have only one unique domain.
R4.6 Data consistency rules	R4.6.1 The data values for a business attribute must be consistent when the attribute is duplicated for performance reasons or when it is stored redundantly for any other reason
	R4.6.2 The duplicated data values of a business attribute must be based on the same domain and on the same data quality rules.

Table B.4 Data validity rules