Performance Evaluation of Blocking Methods for Duplicate Record Detection

By

Raed Mahmmod Al Dummor

A Thesis

Submitted in Partial Fulfillment of the Requirements for the Master Degree in Computer Information Systems

Supervisor

Prof. Musbah M. Aqel

Department of Computer Information Systems
Faculty of Information Technology
Middle East University
Amman, Jordan

July, 2010

AUTHORIZATION FORM

إقرار تفويض

أنا رائد محمود الضمور أفوض جامعة الشرق الاوسط بتزويد نسخ من رسالتي للمكتبات أوالمؤسسات أو الهيئات أو الافراد عند طلبها .

التوقيع :

التاريخ : 2010/8/10

Authorization Statement

I, Raed Mahmmod Al-Dummor, authorize the Middle East University to supply a copy of my thesis to libraries, establishments or individuals upon their request.

Signature: 

Date: 10 /8/2010

Middle East University

Examination Committee Decision

This is to certify that the thesis entitled "**Performance Evaluation of Blocking Methods for Duplicate Record Detection**" was successfully defended and approved on August 10th 2010.

Examination Committee Members **Signature**

Prof. Nidal F. Shilbayeh
Professor, Department of Computer Science , Faculty of Information Technology, Middle East University

Prof. Musbah M. Aqel
Professor, Department of Computer Information Systems, Faculty of Information Technology, Middle East University

Dr. Ahmad H. Al-Omari
Assistant Professor, Department of Computer Information Systems, Faculty of Information Technology, Applied Science Private University

DECLARATION

I do declare hereby that the present research work has been carried out by me under the supervision of Prof. Musbah M. Aqel , and this work has not been submitted elsewhere for any other degree , fellowship or any other similar title.

Signature: 

Date: 10 /8/2010

Raed Mahmmod Al Dummor
Department of Computer Information Systems
Faculty of Information Technology
Middle East University
Amman, Jordan
Email: Dummor@gmail.com
P.O.Box 284 Amman 11512 , Jordan

DEDICATION

This thesis is dedicated to my parents, my wife and partner for life, **Rula**, and our daughter **Yara**, for their understanding and support during the time of this research.

ACKNOWLEDGEMENTS

I am highly indebted to my supervisor Prof. Musbah M. Aqel, Faculty of Information Technology, Middle East University, for his eminent guidance, constant supervision, and whelming encouragement throughout my thesis work.

I am grateful to a number of friends for their moral support, encouragement, and technical discussions.

Finally, I would like to take a moment to thank my family members for their immense patience, emotional support and encouragement during my entire graduate career.

ABSTRACT

Performance Evaluation of Blocking Methods for Duplicate Record Detection

By

Raed Mahmmod Al Dummor

Duplicate record detection process is the process of identifying pairs of records in one or more datasets that refer to the same real world entity (e.g. patient or customer), where these individual entities might be erroneous and incomplete. In addition, there exists no unique identifying key for these entities that would allow to directly identifying them as duplicates. A main challenge when detecting duplicate records is the complexity of the detection process: potentially each record in a dataset has to be compared to all records in the same dataset or another dataset, the number of record pair comparisons grows quadratically with the number of records to be compared. A large variety of methods, collectively known as *blocking methods*, have been developed to deal with this quadratic complexity problem.

Blocking Methods reduce the number potential record pair comparisons by partitioning the datasets into a set of mutually exclusive blocks or clusters using a blocking key (i.e. a single record attribute or a combination of attributes). All records sharing the same blocking key value will be placed in the same block and only records within a block will be compared.

In this thesis, we experimentally compare and evaluate two recently developed, blocking methods, the sorted blocks and standard suffix array and its improvements, with two older methods, the standard blocking and sorted neighborhood blocking within a common framework with regard to the quality of the candidate record pairs generated by them.

The experiments results on synthetic dataset show that sorted neighborhood blocking method outperforms the standard blocking and that sorted blocks slightly outperforms it in terms of accuracy. Also our experiments results show that the accuracy of the improved suffix array method is much higher than the standard suffix array and that standard blocking can be dramatically improved using standard suffix array and its improvements.

الملخص

تقييم أداء أساليب التجزئة في أكتشاف السجلات المكررة

رائد محمود الضمور

اكتشاف السجلات المكررة هو الاجراء الذي يختص بتحديد اية أزواج من السجلات في واحدة أو أكثر من مجموعات البيانات التي تشير إلى نفس الكيان في العالم الحقيقي (مثل المريض أو العميل) ، وبالاضافة الى ذلك قد تكون هذه السجلات خاطئة وغير مكتملة احيانا. كما قد لا يوجد مفتاح واضح لتمييزها كسجلات مكررة .

ان التحدي الرئيسي عند اكتشاف السجلات المكررة هو تعقيد عملية الاكتشاف، حيث يجب مقارنة كل سجل في مجموعات البيانات مع باقي السجلات في نفس المجموعة او المجموعات الاخرى. وبالتالي فأن عدد المقارنات لازواج السجلات ينمو بشكل تربيعي بالمقارنة مع عدد السجلات التي يمكن مقارنتها .ولهذا تم وضع مجموعة كبيرة ومتنوعة من الأساليب التى عرفت بأسم "*أساليب التجزئة*" ، بهدف معالجة هذه المشكلة التربيعية المعقدة .

تهدف اساليب التجزئة الى تقليل عمليات مقارنة ازواج السجلات من خلال تجزئة مجموعات البيانات الى مجموعة من الكتل التبادلية بأستخدام مفتاح للتجزئة (مكون من سمة سجل واحد أو مجموعة من السمات) ، حيث يتم وضع كافة السجلات التي تتشارك في نفس المفتاح في نفس الكتلة ويتم مقارنة السجلات فقط داخل هذه الكتلة.

في هذه الدراسة نقوم ومن خلال التجربة العملية بتقييم ومقارنة اثنين من هذه الاساليب التي وضعت حديثا كأساليب للتجزئة وهي الكتل المرتبة ومصفوفة اللواحق المعيارية وتحسيناتها مع اسلوبين قديمين هما الكتل المعيارية والكتل المتجاورة المرتبة وذلك ضمن إطار عام فيما يتعلق بجودة أزواج السجلات التي يتم انتاجها.

تؤكد نتائج التجارب التى تم اجراءها على مجموعة من البيانات المركبة ان مصفوفة اللواحق المعيارية تتفوق بالاداءعلى الكتل المعيارية في حين تتفوق الكتل المرتبة بشكل طفيف فيما يتعلق بالدقة. كما تؤكد نتائج تجاربنا ان دقة اسلوب مصفوفة اللواحق المحسنة هي اعلى بكثير من مصفوفة اللواحق المعيارية كما ان الكتل المعيارية يمكن تحسينها بأستخدام مصفوفة اللواحق المعيارية وتحسيناتها .

LIST OF TABLES

LIST OF FIGURES

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

For decades there is a tremendous growth for the need of a various application that can access, relate, use, and integrate of multiple disparate information sources and repositories including databases, knowledge bases, file systems, digital libraries and electronic mail systems. All Indications suggest that this growth will continue in the years to come. The contributing factors to this increase are the expansions of business over the globe and the revolution of the fast reliable inexpensive communication links.

Data integration is a core issue in these applications, especially in the area of Business Intelligence (BI), Customer Relationship Management (CRM), Online Analytical Processing (OLAP), data mining and in the area of e-Business. Ensuring data integrity in the process of integration of heterogeneous data sources has proven to be a very challenging task due to the difficulties and challenges faced in dealing with the heterogeneity, technology obsolescence and semantic discrepancy associated with multiple source information systems. (Tang et al., 2003) (Ziegler & Dittrich, 2004)

1.1 Data Duplication and Data Quality

While data integration is essential and important in various domains, we have to be aware that the quality of the information integrated from diverse data sets is highly dependent on the quality of these data sets. A data set that is incomplete, noisy and inconsistent may result in conclusions that do not truly reflect the underlying truth and lead to false decisions. So we need to pre-process data sets to create a complete, clean and consistent data set prior to any data integration tasks. (Bleiholder & Naumann, 2008)

The results of poor quality of data are often experienced in everyday life, but without making the necessary connections to its causes. For example, the late or missed delivery of a letter is often blamed on a dysfunctional postal service, although a closer look often reveals data related causes, typically an error in the address, originating in the address database. Similarly, the duplicate delivery of automatically generated post is often indicative of a database record duplication error. Inaccurate and duplicate addresses are examples of data quality problems. (Scannapieco et al., 2005)

Data quality is a very complex concept, the complexity results from its composition of various characteristics or dimensions but most researchers have traditionally agreed on it to be defined as **fitness for use.** Another definition for data quality is ”**the distance between the data views presented by an information system and the same data in the real world”** such a definition can be seen as an ”operational definition”, although evaluating data quality on the basis of comparison with the real world is a very difficult task. (Bertolazzi et al., 2003)

Although a standard set of dimensions not yet defined, the researcher’s community agrees on a common minimal set of specific quality characteristics. Characteristics that captures a specific aspect of quality and data satisfying those quality characteristics is said to be of high quality. (Müller & Freytag, 2003)

The most commonly referenced dimensions include accuracy, completeness, currency and consistency. These four dimensions are only some of a large set of dimensions proposed in the data quality literature. For instance, many subjective dimensions have also been

proposed to characterize data quality, including, among others, reputation, objectivity, believability, interpretability. (Scannapieco et al., 2005)

1.2 Duplicate Record Detection and Blocking

The problem that we study has been known for decades as the record linkage or the record matching problem in the statistics community where the goal of record matching is to identify a group of records from different sources that refer to the same real world entity. Many recent studies have pointed out that similar types of problem have been described differently in different research fields, for example In the database community, the problem is described as merge-purge, data deduplication which emphasizes the elimination of duplications in a database. Also the same problem is described as instance identification, database hardening and name matching in the Artificial Intelligence community. The names coreference resolution, identity uncertainty, and duplicate detection are also commonly used to refer to the same task. We will use the term duplicate record detection in this thesis or simply duplicate detection. (Aizawa & Oyama,2005), (Elmagarmid et al., 2007)

Duplicate record detection can be defined as" **the process of identifying different or multiple records that refer to one unique real world entity or object"**. Typically, the process of duplicate detection is preceded by a data preparation stage during which data entries are stored in a uniform manner in the database, resolving (at least partially) the structural heterogeneity problem. The data preparation stage includes a parsing, a data transformation, and a standardization steps in order to transform the data present in the sources into a common representation. In this stage, many problems arise such as naming conflicts happened when the same name is used for different objects (homonyms) or

different names are used for the same object (synonyms). For example, surname may be used to represent last name in one data set, while family name may be used to represent last name in another data set. Also, state and district may be used to describe addresses in different data sets. (Bleiholder & Naumann, 2008)

The approaches that deal with data preparation are also described under using the term ETL (Extraction, Transformation, Loading). These steps improve the quality of the in-flow data and make the data comparable and more usable. (Gu et al., 2003)

After the data preparation phase, the data are typically stored in tables having comparable fields. The next step is to identify which fields should be compared. For example, it would not be meaningful to compare the contents of the field surname with the field Address. Even after parsing, data standardization, and identification of similar fields, it is not trivial to match duplicate records. Misspellings and different conventions for recording the same information still result in different, multiple representations of a unique object in the database. For example, in the bibliographic data sets, one may store the first name and last name of an author (e.g. "Musbah Aqel"), while another data set may store only the initials and the last name of the person (e.g. "M. M. Aqel") and simply integrating them without eliminating these duplicates will result in data with redundancy. (Elmagarmid et al., 2007)

The duplicate detection methods have its roots from the record linkage theory and share many techniques used to solve the same problems. It's worth highlighting the main phases of a record linkage process that is applicable to duplicate detection process where major phases include as shown in Figure 1.1: (Cibella et al., 2008)

1. Pre-processing of the input files

2. Choice of the identifying attributes (matching variables)

3. Choice of the comparison function

4. Creation of the search space of link candidate pairs

5. Choice of the decision model

6. Selection of unique links

7. Record linkage evaluation

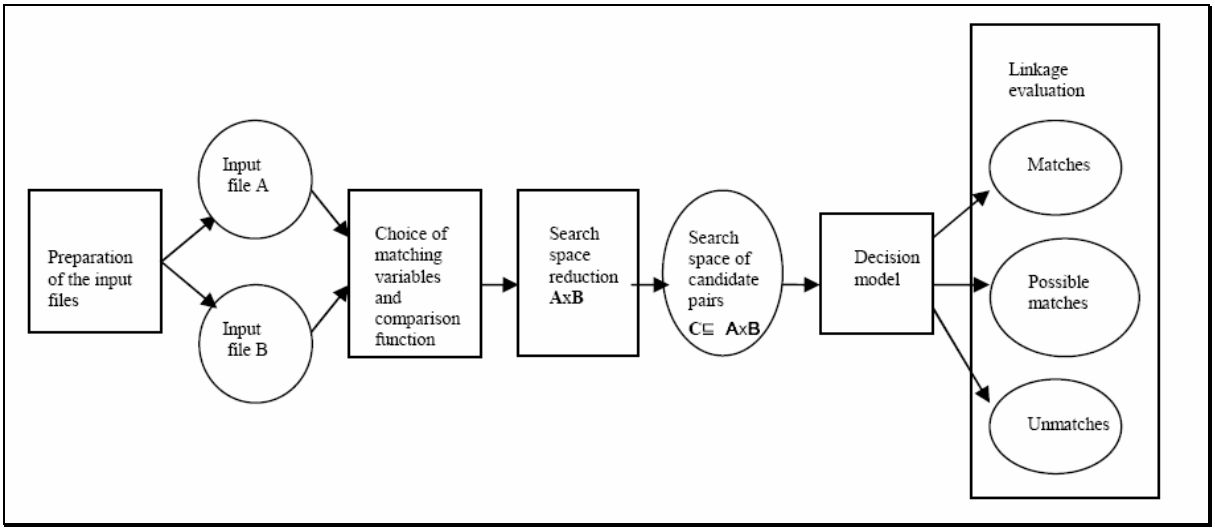

Figure 1.1: Phases of a record linkage project (Cibella et al., 2008)

One of the most important stages in any record linkage and duplicate record detection process is the reduction of the search space of the candidate pairs. To reduce the large amount of potential record pair comparisons, traditional data linkage techniques employ blocking where a single record attribute or a combination of attributes called the blocking key value is used to split the databases into blocks. All records having the same value in the blocking key will be inserted into one block, and candidate record pairs are then generated only from records within the same block. While the aim of blocking is to reduce the number of record pair comparisons made as much as possible (by eliminating pairs of

records that obviously are not matches), it is important that no potential match is removed by the blocking process. The candidate record pairs generated by the blocking process are then compared using a variety of comparison functions applied to one or more (or a combination of) records attributes. And finally a decision is made to classify each pair as matches, unmatched and possible matches. (Christen , 2007)

1.3 Motivation

The revolution and advances in the field of database systems and computer networks open the door widely for the need to access and manage information from a variety of sources and applications using different data models, representations and interfaces has created a great demand for tools supporting data and systems integration. One reason for this need was the paradigm shift from centralized to client-server and distributed systems, with multiple autonomous sources producing and managing their own data. A more recent cause for the interest in integration technologies is the emergence of E-Commerce and its need for accessing repositories, applications and legacy systems located across the corporate intranet or at partner companies on the Internet.

The motivation for the work in this thesis mainly comes from my work for many years as a programmer and database administrator which have encourage me to focus on the integration of multi-source information systems. And one of the major motivation for this is the need to find the best and optimal method for detecting duplicates data in such systems in my enterprise in order to provide a comprehensive view of data represented in different ways and to help the users of such systems to automate the process of duplicate detection

so that they don't have the need to directly interact and manually deal with the underlying data.

1.4 Problem Definition

Many organizations and businesses collect vast amounts of data. The need for techniques that allow efficient processing and analyzing of such data is essential and important in such projects. One of the most important stages in this process is the pre-processing stage where the detecting and removing of duplicate records that relate to the same entity within one database. Similarly, linking or matching records relating to the same entity from several databases is often required as information from multiple sources needs to be integrated, combined or linked in order to enrich data and allow more detailed data mining analysis. The aim of such integration is to match and aggregate all records relating to the same entity, such as a patient, a customer, a person.

Duplication detection can be used to improve data quality and integrity to allow re-use of existing data sources for new studies, and to reduce costs and efforts in data acquisition.

A major challenge to achieve such goals is that many systems depend on the accuracy of databases to carry out operations. Therefore, the quality of the information (or the lack thereof) stored in the databases can have significant cost implications to a system that relies on information to function and conduct business. In an error-free system with perfectly clean data, the construction of a unified view of the data consists of linking or joining two or more records on their key fields. Unfortunately, data often lack a unique, global identifier that would permit such an operation.

Efficiency is another major challenge in duplicate detection especially when the data sets are large. In real life data sets, most record pairs are not duplicates, and comparing them will waste plenty of time and resources .To reduce the large amount of potential record pair comparisons, a number of blocking methods have been proposed to reduce the number of comparisons in order to improve the efficiency whilst still maintaining accuracy.

These methods needed to be evaluated for quality, efficiency and performance within a common framework, in order to answer questions such as: How does the blocking method improve the quality of the resulting candidate record pairs? Which blocking method performs best for databases with certain characteristics? Which methods have the highest performance for a given resource?

1.5 Contribution

Given these requirements and challenges, we proposed a framework for duplicate record detection that takes its roots from generic designs suggested and used in solving record linkage and duplicate detection problems. Also, we evaluated the performance of the major methods used in the blocking stage which is part of any duplicate record detection process. Below are the contributions we make in this thesis.

- We have surveyed the duplicate record detection problem and introduced some of the latest advances in the recent years, we briefly discussed the major steps in the duplicate detections process and take a closer look of data quality and how it's related to duplicate detection problem. We presented the major methods used for matching records attributes and the strings similarity functions used .We presented and classified the decision models used in identifying duplicates records.

- We have surveyed the major blocking methods used in the blocking process and introduced some of the new methods being used. We presented and classified the major methods used to improve the efficiency using blocking and the measurement tools used to measure its quality and efficiency.
- We experimentally implemented two recently developed, blocking methods, the sorted blocks and standard suffix array and its improvements, with two older methods, the standard blocking and sorted neighborhood blocking using the same data set and the same decision model and evaluated the results of these methods using the standard measurements tools.

1.6 Thesis Outline

The remainder of this thesis is organized as follows. In chapter 2, we review the most important literature related to the duplicate record detection. In chapter 3, we describe the main elements of our framework design for duplicate record detection problem. In chapter 4, we present the conducted experiments with their used settings and their results. Finally, the conclusions and future work are presented in chapter 5. Our thesis includes also references and appendices.

CHAPTER 2

LITERATURE REVIEW

In this chapter, we present a comprehensive review of the literature on duplicate record detection. In Section 2.1, we review several blocking methods that are used to improve the efficiency and scalability of approximate duplicate detection methods. In Section 2.2, we review the similarity measures that are commonly used to detect similar attributes values. In Section 2.3, we review decision models that are used for matching records with multiple attributes. In Section 2.4, we present a range of tools that are used for duplicate detection. Finally in Section 2.5, we presented some of the most important related studies in the field of duplicate record detection.

2.1 Blocking Methods

Efficiency is a major challenge in duplicate detection especially when the data sets are large. In real life data sets, most records pairs are not duplicates, and comparing them will waste plenty of time and resources and similarly for data linkage if unique entity identifiers (or keys) are available in all the databases to be linked, then the problem of linking at the entity level becomes trivial: a simple database join is all that is required. (Christen , 2007)

However, in most cases no unique keys are shared by all records and for two data sets, A and B, are to be linked, potentially each record from A has to be compared with all records from B. The total number of potential record pair comparisons thus equals the product of the size of the two databases, $|A| \times |B|$. With $|A|$, $|B|$ denoting the number of records in the data set. For example, linking two databases with 100,000 records each would result in 10^{10}

(ten billion) record pair comparisons. On the other hand, the maximum number of true matches corresponds to the number of records in the smaller database (assuming there are no duplicate records in the databases, and one record in database A can only match to one record in database B, and vice versa). Thus, the space of potential links becomes sparser when linking larger data sets, while the computational efforts increase exponentially. (Christen & Churches, 2005)

Similarly, when detecting duplication in a data set A, the total number of possible record pair comparisons is |A| x (|A| -1)/2. The performance bottleneck in a duplicate detection process is usually the expensive detailed comparison of fields (or attributes) between pairs of records, making it unfeasible to compare all pairs when the databases are huge. Therefore, while the computational efforts increase quadratically, the number of potential true matches only increases linearly for deduplication, where the number of duplicate records is always less than the number of records in a data set. (Christen, 2007)

To reduce the large amount of possible record pair comparisons, a number of methods have been proposed to reduce the number of comparisons in order to improve the efficiency and at the same time still maintain accuracy. These methods group the data set into blocks or clusters of records where a single record attribute or a combination of attributes called the blocking key value (BKV) is used to split the databases into blocks with respect to a defined criterion or some threshold values. All records having the same value in the blocking key will be inserted into one block, and candidate record pairs are then generated only from records within the same block. (Baxter et al., 2003) ,(Christen,2007) ,(Tamilselvi & Saravanan, 2009)

Blocking is essential in order to make duplication detection possible at all, and to improve the efficiency of the duplication detection process. Blocking, however, will reduce the detection quality, as it is likely that some true matched record pairs will be removed by the blocking process, if records are not being inserted into the correct block (or blocks) due to variations and errors in their BKVs. The blocking process can be split into the following two steps.

1. **Build**: All records from the data set are read, the blocking key values are created from record attributes, and the records are inserted into an index data structure. For most blocking methods, a basic inverted index can be used. The blocking key values will become the keys of the inverted index, and the record identifiers of all records that have the same blocking key value will be inserted into the same inverted index list. (Christen ,2007)
2. **Retrieve**: Record identifiers are retrieved from the index data structure block by block, and candidate record pairs are generated. Each record in a block will be paired with all other records in the same block. The generated candidate record pairs are then compared and the resulting vectors containing the numerical comparison values are given to a classifier.

We should note that the choice of blocking keys is usually done manually by domain and duplication detection experts and are usually chosen to be very general in order to produce a high quality result, while also producing a reasonable reduction in the amount of data required to compare against for each record to be matched. Only recently have learning approaches been explored that aim to automate this step. (De vries et al., 2009)

A blocking key can be defined as either the values taken from a single record attribute (or parts of values, like the first two initial letters only), or the concatenation of values (or parts of them) from several attributes. To reduce the effect of variations, and typographical and other spelling errors, phonetic encodings, such as Soundex, are commonly used when blocking keys are generated. They work by encoding names such that similar sounding values are replaced by the same code. (Christen , 2007)

Next we present some of the major blocking methods that are used to improve the efficiency and scalability of duplicate detection process.

2.1.1 Standard Blocking Method

The Standard Blocking (SB) method group records into disjoint blocks where they have the same blocking key value and then compare all pairs of records only within each block. A blocking key is defined to be composed from the record attributes of the data set. Assuming a data set with N records needed to be deduplicated, and the blocking method resulted in B blocks (all of the same size containing N/B records), the resulting number of record pair comparisons is O (N^2/B). Thus, the overall number of comparisons is greatly reduced. This is of course the ideal case, hardly ever achievable with real data. Thus, the number of record pair comparisons can be dominated by the largest block. (Baxter et al., 2003)

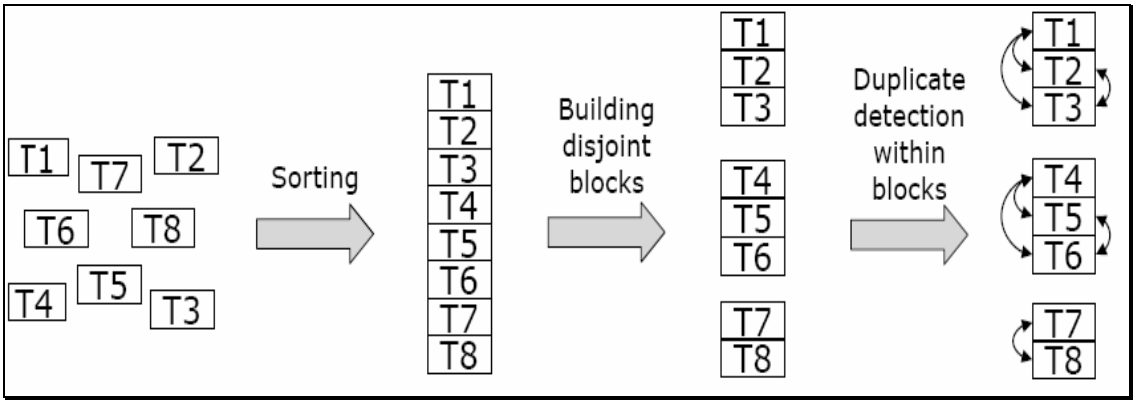

Figure 2.1: Standard blocking method (Draisbach & Naumann, 2009)

An important decision for the blocking method is the choice of a good partitioning predicate or blocking key, which determines the number and size of the blocks. They should be chosen in a manner that potential duplicates appear in the same block. For example in Customer Relationship Management (CRM) applications a typical blocking is by zip-code or by the first few digits of zip-codes. If two duplicate records have retained the same zip code, they appear in the same block and thus can be recognized as duplicates. Other partitioning might be by last name or some fixed-sized prefix of them, by street name, by employer, etc. (Draisbach & Naumann, 2009)

In general, blocks of roughly same size are preferable. For simplicity we use in our work blocks of an equal size as shown in Figure 2.1. In Appendix 1.a we listed our implementation algorithm for standard blocking method.

Although blocking can substantially increase the speed of the comparison process, it can also lead to an increased number of false mismatches due to the failure of comparing records that do not agree on the blocking field. It can also lead to an increased number of missed matches due to errors in the blocking step that placed entries in the wrong blocks, thereby preventing them from being compared to actual matching entries. One alternative is

to execute the duplicate detection algorithm in multiple runs, using a different field for blocking each time. This approach can substantially reduce the probability of false mismatches, with a relatively small increase in the processing time. (Elmagarmid et al., 2007)

Also other major drawbacks of standard blocking are the sizes of the blocks generated. As these sizes depend upon the frequency distributions of the blocking key values, it is difficult to predict the total number of candidate record pairs generated by this blocking technique. (Christen , 2007)

2.1.2 Sorted Neighborhood Method

Sorted Neighborhood Method (SNM) or Windowing method is slightly more elaborate than standard blocking method. The SNM is proposed by Hern´andez and Stolfo to effectively reduce the number of comparisons by limiting the similarity measures on a small portion of the data sets. The method can be summarized in three steps. (Hernández & Stolfo, 1998)

1- **Create key:** A blocking key for each record in the data set is computed by extracting relevant attributes, or a sequence of substrings within the attributes, chosen from the record in an ad hoc manner. Attributes that appear first in the key have a higher priority than those that appear subsequently. For example the key may composed from first three constants characters of a person surname concatenated by the first three constants characters of a person given name and first 3 digits of the social security id.

Given name	Surname	Address	Social security id	Key
Sarah	Bright	Hardman Street	456769608828987	BRGSRH456

2- **Sort data:** The records in the data set are sorted by using the blocking key value created in the first step.

3- **Merge:** A window of fixed size w >1 is moved through the sequential list of records in order to limit the Comparisons for matching records to those records in the window. If the size of the window is w records, then every new record that enters that window is compared with the previous w - 1 record to find "matching" records. The first record in the window slides out of it as shown in Figure 2.2.

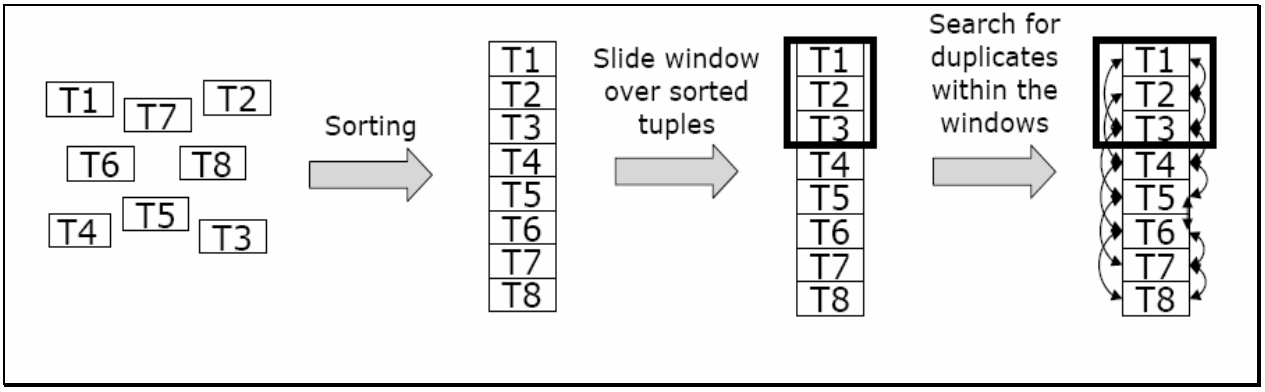

Figure 2.2: Sorted neighborhood method (Draisbach & Naumann, 2009)

For example using zip-codes as blocking key in the Sorted-Neighborhood method, we sort the data according to the zip-code, and then slide a window of fixed sized across the sorted data and compare pairs only within the window. The use of the window limits the number of possible record pair comparisons for each record to w −1. The resulting total number of record pair comparisons for a data set with n records using the sorted neighborhood method is O (wn). In Appendix 1.b we listed our implementation algorithm for Sorted Neighborhood Method.

We can see that the accuracy of the SNM method depends on the quality of the keys chosen. A poorly chosen key will filter out a lot of true duplicates. Also, the size of the window is very important. For records with a large number of duplicates, a small window size will result in the same result as that of a poorly-chosen key, and a large window size will bring in too much unnecessary comparisons. The size of the window (typically between 10 and 20) represents the trade-off between efficiency and effectiveness; larger windows yield longer run times but detect more duplicates. (Draisbach & Naumann, 2009)

To solve this problem, Hernandez and Stolfo implemented a multi-pass sorted neighborhood method which Execute the SNM method independently several times and each time the records are sorted using a different key and a small window size. Finally the result is computed by merging the results from the multi-pass. (Hernández & Stolfo, 1998)

2.1.3 Sorted Blocks Method

Draisbach & Naumann presented a new generalized algorithm, the Sorted Blocks method that generalized the blocking algorithm and slightly improves upon it in terms of efficiency for detecting duplicates against the overall number of comparisons. (Draisbach & Naumann, 2009)

The basic idea of the Sorted Blocks method is to sort all records so that duplicates are close in the sort sequence, then partition the records into disjoint sorted subsets, and finally to overlap the partitions. The size of the overlap can be defined using u, e.g., u = 3 means that three records of each neighboring partition are part of the overlap, which hence has a total size of 2u. Within the overlap, a fixed size window with size u + 1 is slide across the sorted data and all records within the window are compared. In this way, the additional

complexity of the overlap is linear. Note that this windowing method is used only in the overlapping part; within a partition all pairs of records are compared.

Figure 2.3 shows the Sorted Blocks method for partitions with equal size and an overlap u = 2. Within a partition, each record is compared with all other records. The overlap between the partitions results in several windows, which have to be checked for duplicates. For instance, overlap between partitions P1 and P2 has windows F(P1;P2) and between partitions P2 and P3 has windows F(P2;P3) .

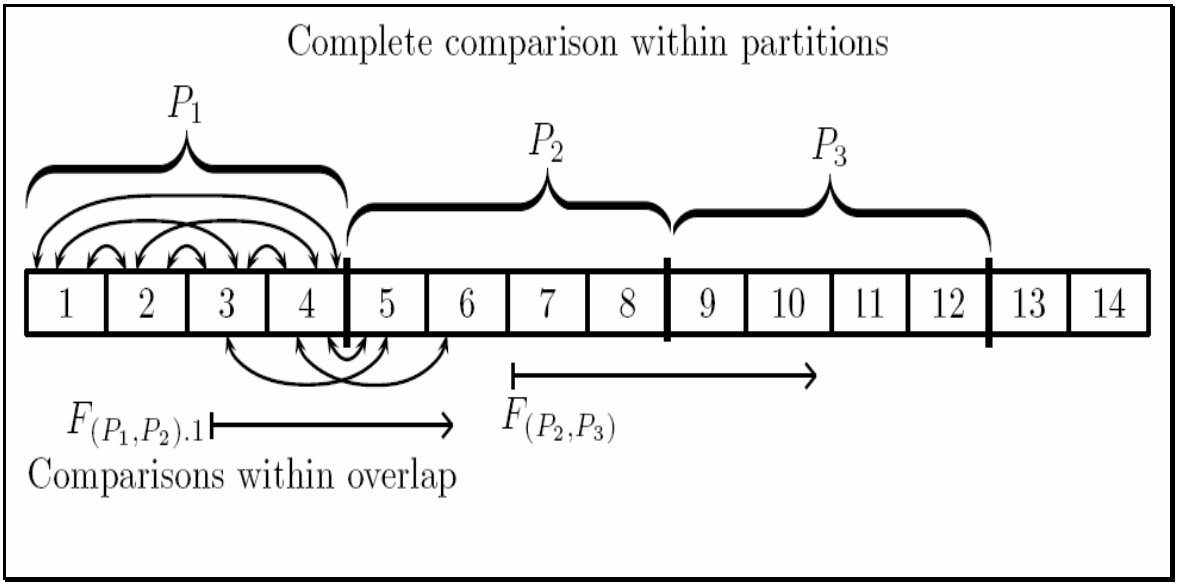

Figure 2.3: Sorted blocks method.

The sorted blocks parameterize the degree of overlap from none (Blocking method) to w-1 (Sorted Neighborhood method). In Appendix 1.c we listed our implementation algorithm for the Sorted Blocks Method using equal block sizes with an overlap partition equal to the half block size.

2.1.4 Standard Suffix Array Blocking Method

The Suffix Array blocking method proposed by Akiko Aizawa and Keizo Oyama as a fast and efficient domain independent method for blocking of large scale record linkage for multi-source information integration. (Aizawa & Oyama , 2005)

The basic idea is to insert the blocking key values (BKV) and their variable length suffixes into a suffix array based inverted index (an indexing structure). The suffix array contains strings and their suffixes in an alphabetically sorted order. The purpose of the indexing structure is to find a set of references to original records that contain a certain suffix, when queried with that suffix. (Christen , 2007)

In this blocking technique, only suffixes down to a minimum length min_suff_length are inserted into the suffix array. For example, for a blocking key value `Christen' and a minimum length min_suff_length = 3, will be decomposed into the following suffixes values `Christen', `hristen', `risten', `isten', `sten', and `ten', then it will be inserted into the suffix array (along with the identifiers of all records that have this BKV will be added to the corresponding inverted index lists). Similar to Q-gram based and canopy clustering blocking, the identifier of each record will be inserted into several blocks (i.e. inverted index lists), according to the length of its blocking key value (a blocking key value of length c character will be inserted into c - min_suff_length + 1 index lists).

After generating suffix array from its BKVs, and inserting the suffixes from these suffix array into the indexing structure, one further optimization step is carried out in order to limit the maximum block size, only values in the suffix array inverted index that have less than a maximum number block of record identifiers in their corresponding list will be used.

So they introduced the Max_block_size parameter for this purpose to deal with low values of minimum suffix length especially when some words may all feature a common suffix.

This occurrence can result in the block for common suffixes such as ‘ing’ to be extremely large, and this has a significant adverse effect of the efficiency of the suffix array blocking method. Therefore, a basic rule is introduced to remove any particular block entirely if it contains references of more than Max_block_size . For example, if the suffix array value `ten' appears in 20 records, and the value `sten' in only 5 records, and Max_block_size = 6, then `ten' is considered to be too general and is not used in the retrieve step (when blocks are extracted and record pairs are generated), as it would produce too many pairs. (De vries et al., 2009)

The Suffix Array blocking method retains accuracy by allowing the correct blocking of records that share common rare suffixes, even if they are short, while excluding very common suffixes. Since each input BKV is decomposed into multiple suffixes, the removal of the block for some of these suffixes does not adversely affect the result. Another advantage of suffix array blocking over traditional blocking is that it is not prone to blocking key value errors. If errors occur in the BKVs, usually not all of the suffixes of these BKVs will change, only some of the longer ones. One record will be inserted into several blocks, adding a form of redundancy to try to ensure that true matched record pairs are grouped into the same block at some stage.

Analysis of the Suffix Array blocking against many recent alternatives found that the efficiency gain is very high for this method, but the accuracy can suffer with standard data

sets and when the blocking key value is chosen by concatenating several key fields, as is the standard for comparison.

2.1.5 Improved Suffix Array Blocking Method

Timothy de vries and others have proposed some improvements to enhance the limitations of the suffix array blocking method such as when two BKV substrings may be long enough to surpass the minimum suffix length parameter, and contain just one small difference due to misspellings or typographical errors, resulting in an inability to group these two BKVs into the same block. For example consider the BKVs 'tophills' and 'topbills' with a minimum suffix length min_suff_length = 5 , the suffix array will be 'hills', 'phills', 'ophills', and 'tophills' for the first BKV, and 'bills', 'pbills', 'opbills', and 'topbills' for the second. Clearly, most of these suffixes have a high similarity. However, the standard suffix array blocking method will not group any of these suffixes into the same block, causing it to miss the comparison that should be carried out between these two BKVs. (De vries et al., 2009)

De vries propose an approach towards solving this problem, by carrying out a grouping operation on similar suffixes using many methods for grouping or clustering of these suffixes. However we should consider the time complexity of the indexing method in order to avoid an overall scalability decrease for duplication detection such as the large number of comparisons between the BKV suffixes.

A simple method for grouping that does not cause adverse scalability reductions can be implemented by only checking nearby neighbors when carrying out the grouping. Specifically by implementing a method that can iterate over the sorted set of BKV suffixes,

comparing each current suffix with the following one using a similarity metric to compare BKV suffixes such as edit distance , Q-grams or Longest Common Subsequence (LCS) operator. If a similarity exists, the following suffix can be merged into the same group as the current suffix. If no similarity exists, the method continues with the next suffix becoming the one currently being processed.

Appendix 1.d listed our implementation algorithm for the suffix array using similarity functions for suffix array comparison.

2.1.6 Q-gram Based Blocking Method

This method aims to allow blocking such that variations in the blocking key values (like deletions, insertions or substitutions of characters) do not affect the blocking process. It works by inserting records into more than one block. This is achieved by transforming the blocking key values into lists of Q-grams (sub-strings containing q characters), and creating all combinations of sub-strings down to a certain length (determined by a threshold value t between 0.0 and 1.0 which designates the fraction of the shortest sub-lists to be generated relative to the length of the Q-gram list). The resulting Q-gram sub lists are sorted and inserted into an inverted index, which will be used to retrieve the corresponding record numbers in a block. (Gu &Baxter , 2004) , (Christen , 2007)

For example, assume a blocking key value `peter', q =2 and a threshold value of t = 0.8. The 2-gram list for this value is [`pe',`et',`te',`er'] with four elements, and using the threshold 0.8 results in 4 x 0.8 = 3:2, rounded to 3, which means all sub-list combinations with a length of 3 are generated: [`et',`te',`er'], [`pe',`te',`er'], [`pe',`et',`er'],and [`pe',`et',`te']. Therefore, the record identifiers of all records with blocking key value `peter' will be

inserted into five inverted index lists (the original 2-gram list is also used as inverted index key) with key values `peetteer', `etteer',`peteer', `peeter', and `peette'.

The number of sub-lists created for a blocking key value depends both on the length of the key value and the chosen threshold. The lower the threshold the shorter the sub-lists and therefore many different inverted index key values. But also the more sub-lists there will be per blocking key value result in smaller blocks in the inverted index. (Baxter et al., 2003)

Like standard blocking, the number of record pair comparisons with two data sets with N records each and B blocks all contain the same number of records is O (N^2/B). However, the number of blocks b will be much larger in Q-gram blocking. For a blocking key value of length n characters, there will be (n - q + 1) Q-grams, and therefore n sub-lists containing (n - 1) Q-grams, n x (n - 1) sub-lists containing (n - 2) Q-grams, etc. This explosion in the number of sub-lists limits Q-gram based blocking to short blocking key values. (Christen , 2007)

2.1.7 Canopy Clustering Method

Canopies are a two-step efficient clustering method for high dimensional data sets that have been proposed for speeding up duplicate detection process. The key idea behind this is to use a cheap similarity measure such as Jaccard to roughly partition the whole data set into a number of overlapping clusters or subsets which are referred to as canopies and then employ standard similarity measures to compare records within the same canopy. For attributes with text values, they use an inverted index to efficiently construct canopies. This method is also used for reducing the computational cost. (Elmagarmid et al., 2007)

Others also propose the use of Canopy Clustering with TFIDF (Term Frequency/Inverse Document Frequency) forms blocks of records based on those records placed in the same canopy cluster. A canopy clusters formed by choosing a record at random from a candidate set of records (initially, all records) and then putting in its cluster all the records within a certain loose threshold distance of it. The record chosen at random and any records within a certain tight threshold distance of it are then removed from the candidate set of records. (Baxter et al., 2003)

Both Jaccard or TFIDF/cosine similarities are based on Q-grams (or more generally, tokens) and can be implemented efficiently using an inverted index with the Q-grams. Where the blocking key values are first converted into Q-gram lists and then each Q-gram is inserted into an inverted index. For TFIDF/ cosine similarity additional information has to be calculated: for each unique Q-gram the number of records that contain the Q-gram, i.e. its term frequency (TF); and within the inverted index the document frequency (DF) for each Q-gram in each record (i.e. the frequency of a Q-gram in a blocking key value). Once all records in a database have been read and processed, the TF and DF values can be normalized and the inverse document frequency (IDF) can be calculated for each Q-gram. No such frequency information or normalization is required for Jaccard similarity. (Christen , 2007)

When record identifiers are retrieved from the inverted index they are inserted into a pool of candidate records. Canopy clusters are then generated by randomly selecting a record from the pool (which will become the centroid of the cluster), and adding all records from the pool into the cluster that are closer than a loose similarity value threshold t_{loose}. Of

these, all records within a tight similarity threshold t_{tight}, with $t_{tight} >> t_{loose}$ are removed from the candidate pool of records (these are the records that have the most similar blocking key values to the centroid record). This process is repeated until no candidate record is left in the pool.

Using the Jaccard measure, the similarity between two records is calculated as the number of Q-grams in the two blocking key values in common is divided by the union of Q-grams in the two values. Where the TF-IDF/cosine similarity is calculated in a related way, but additionally TF and IDF values are included, which makes the calculations computationally more expensive. If both thresholds t_{loose} and t_{tight} are set to a value of 1.0 (i.e. only exact similarity), the canopy clustering method reduces to standard blocking.

Similar to the previously described blocking methods, the canopy clustering approach with global thresholds t_{loose} and t_{tight} will result in blocks (i.e. canopy clusters) of different sizes (even though the TF-IDF/cosine similarity measure to some degree adjust similarity weights according to the frequency of the Q-grams in the blocking key values). The number of record pair comparisons resulting from canopy clustering is $O(fn^2/c)$ where n is the number of records in each of the two data sets, c is the number of canopies and f is the average number of canopies a record belongs to. The threshold parameter should be set so that f is small and c is large, in order to reduce the amount of computation. However, if f is too small, then the method will not be able to detect typographical errors. (Baxter et al., 2003), (Christen, 2007)

2.1.8 String Map Based Blocking Method

This method is originally based on the idea of mapping the blocking key values (assumed to be strings) to objects in a multi-dimensional Euclidean space, such that similarities (or distances, like edit distance) between pairs of strings are preserved followed by finding pairs of objects in this space that are similar to each other. Others have modified the FastMap algorithm into StringMap, which has a linear complexity in the number of strings to be mapped. (Christen , 2007)

This algorithm iterates over d dimensions; for each it finds two pivot strings and then forms orthogonal directions and calculates the coordinates of all other strings on these directions.

In the second step, the authors use R-trees as multidimensional data structure in combination with a queue to efficiently retrieve pairs of similar strings. Choosing an appropriate dimensionality d is done using a heuristic approach that tries a range of dimensions and selects the one that minimizes a cost function (dimensions between 15 and 25 typically seem to achieve good results) .In implementation, the replacement of the R-tree data structure with a grid based index, as most tree-based multidimensional index structures degrade rapidly with increasing dimensionality. It is reported that with more than 15 to 20 dimensions, in most tree based indices all objects in an index will be accessed when performing similarity searches.

The grid based index works by having a regular grid of dimensionality d implemented as an inverted index in each dimension (i.e. all objects mapped into the same grid cell in a dimension are inserted into the same inverted index list). The Retrieve step is implemented in a similar way as canopy clustering described above. An object (blocking key value) is

randomly picked from the pool of (initially all) objects, and the objects in the same, as well as in the neighboring grid cells, are then retrieved from the index. Similar to canopy clustering, two thresholds t_{loose} and t_{tight} are used to put objects into clusters.

This blocking method can be modified by replacing the global thresholds with nearest-neighbor parameters n_{loose} and n_{tight} in the same way as described with canopy clustering.

2.2 Attribute Similarity Measures

Given a pair of records, most traditional duplicate detection approaches employ generic string similarity measures or domain specific measures to compute the attribute similarity and make decisions based on the similarity score. Typographical variations of string data is among the most common sources of mismatches in database entries therefore, duplicate detection typically relies on string comparison techniques to deal with these variations.

In addition to the typographical variations in attribute values the data entry operations will determine the types of errors and their distribution such as: (Christen, 2006)

- Handwritten forms are scanned and optical character recognition (OCR) is applied the most likely types of errors will be substitutions between similar looking characters (like ‘q’ and ‘g’), or substitutions of one character with a similar looking character sequence (like ‘m’ and ‘r n’, or ‘b’ and ‘l i’).
- Manual keyboard based data entry will mainly result in wrongly typed neighboring keys (for example ‘n’ and ‘m’, or ‘e’ and ‘r’).
- Data entry over the telephone (for example as part of a survey study) is a confounding factor to manual keyboard entry. The person doing the data entry might not request the

correct spelling, but rather assume a default spelling (which is based on the person's knowledge and cultural background).

- Limitations in the maximum length of input fields can force people to use abbreviations, initials only, or even disregard some parts of a name.
- People themselves sometimes report their names differently depending upon the organization they are in contact with, or deliberately provide modified or wrong names.

In addition to string similarity measures there are multiple methods have been developed for the task of approximate string matching, and each method works well for particular types of errors. Table 2.1 shows some of the available functions used during the comparison process.

Exact string	either field value strings are the same or not
Truncated string	only consider beginning of strings
Approximate string	using Jaro, Winkler, Edit distance, Bigram etc. algorithm
Keying difference	allow a certain number of different characters
Encoded string	using Soundex, NYSIIS, Phonex etc. algorithm
Date	allow day tolerance
Time	allow minute tolerance
Numeric absolute	allow absolute tolerance
Numeric percentage	allowing percentage tolerance
Age	allow percentage tolerance
Distance	allow kilometer tolerance, for example for postcode cancroids

Table 2.1: Available attribute comparison functions (Christen & Churches, 2005)

In this thesis we will focus on string similarity measures that basically can be classified into three categories: Character-Based Similarity Metrics, Token-Based Similarity Metrics, and Phonetic Similarity Metrics. (Elmagarmid et al., 2007)

Next, we describe each category of these metrics and give some examples of them.

2.2.1 Character-Based Similarity Metrics

Character-level similarity measures match strings character by character. The edit distance is one of the most commonly used measures for string similarity. It models the conversion from one string to the other in three operations: insertion, deletion, and substitution. The edit distance between two strings is then given by the minimum number of operations required to transform one string to the other. Waterman generalizes the edit distance by allowing multiple deletions and insertions to handle strings that are either truncated or shortened. Furthermore, for strings with mismatches at the beginning and at the end, Smith and Waterman extend the edit distance by assigning lower costs to the characters at the beginning and the end of a string than in the middle because they observe that it is less likely to make mistakes in the middle of a string than at the beginning and the end. (Cohen et al., 2003) (Elmagarmid et al., 2007)

2.2.1.1 Levenshtein Edit Distance

Levenshtein edit distance is named after the Russian scientist Vladimir Levenshtein, who devised the algorithm in 1965. It is defined as the minimum number of edit operations required to change one string S_1 into another S_2 (not necessarily of the same length). It is calculated using a dynamic programming algorithm. (Elmagarmid et al., 2007) .The edit distance metrics work well for catching typographical errors, but they are typically ineffective for other types of mismatches.

There are three types of edit operations: (Wang, 2008)

- Insert a character into the string,
- Delete a character from the string,

- Substation or replacing one character with a different character.

For Example consider S_1= quickly , S_2= qucehkly

The edit distant between s_1 and s_2 is denoted by Ed (S_1, S_2) =3

There are multiple answers for their transformation, one of them as follow ,the deletion of i from S_1 ,the insertion of e after c in S_1 and the insertion of h before k in S_1 .

Each one the above transformations are called an alignment and represent a possible explanation of the error made.

The time-complexity of the edit distance algorithm is O ($|S_1|*|S_2|$), where $|S_1|,|S_2|$ are the string length. I.e. O (n^2) if the lengths of both strings are about `n'. (Elmagarmid et al., 2007)

Figure 2.4 shows the algorithm used for a the edit distance that takes two strings, S_1 of length m, and S_2 of length n, and computes the edit distance between them: (Black, 2008)

```
Int  LevenshteinDistance(char s1[1..m], char s2[1..n])
 {    // d is a table with m+1 rows and n+1 columns
      declare int d[0..m, 0..n]
       for i from 0 to m
             d[i, 0] := i // deletion
            for j from 0 to n
                d[0, j] := j // insertion
               for j from 1 to n
                    {      for i from 1 to m
                          {       if s1[i] = s2[j] then
                                     d[i, j] := d[i-1, j-1]
                                 else
                                   d[i, j] := minimum  (  d[i-1, j] + 1,   d[i, j-1] + 1 ,
                                           d[i-1, j-1] + 1    )
                           }
                     }
     return d[m, n] }
```

Figure 2.4: Levenshtein Edit Distance algorithmes

The edit distance similarity metric between two strings is constructed ranging from 0 to 1.0 using a normalized formula. (Wang, 2008)

$$\mathbf{Sim}_{\mathbf{Edit\ distance}\ (S_1,S_2)} = 1 - \frac{\mathbf{ED}(S_1,S_2)}{\mathbf{Maxlen}(S_1,S_2)}$$

Where Maxlen is the maximum length of the two strings and ED is the edit distance between S_1 and S_2. (i.e. minimum number of edit operations required to change one string S_1 into another S_2) , where a similarity value of 1.0 means that that the two strings are exactly the same , and zero indicates a little similarity . Table 2.2 provides more examples of similarity values obtained using the edit distance algorithms for several pairs of surnames.

S_1	S_2	$ED(S_1,S_2)$	$Sim(S_1,S_2)$
chamberlain	chambernain	1	0.909
chilcrott	childott	2	0.777
armqsorqng	armstcrong	4	0.600
verdouw	werduow	3	0.571
dixon	dickson	3	0.571

Table 2.2: Examples of similarity values obtained using edit distance algorithm

2.2.1.2 Jaro Distance Metric

Jaro introduced a string comparison algorithm that was used for comparison of first and last names, and mainly takes into account typical spelling deviation for measurement of closeness of two strings for comparison. Jaro string distance metric accounts for insertion, deletion and transpositions. (Cohen et al., 2003)

The basic algorithm for computing the Jaro metric for two strings S_1 and S_2 includes the following steps:

1- Compute the string lengths for both S_1 and S_2.

2- Find the number of common characters in the two strings; the definition of common is that the agreeing character must be within ½ the length of the shorter string. (i.e. S_1 (i) = S_2(j) and $|i-j| \leq \min\{ |S_1|,|S_2|\}/2$)

3- Find the number of transpositions. The definition of transposition is that the character from one string is out of order with the corresponding common character from the other string. We compare the *i*th common character in S_1 with the *i*th common character in S_2, each no matching character is transposition.

The Jaro Distances similarity metric between two strings is constructed ranging from 0 to 1.0 using a normalized formula.

$$\mathbf{Sim_{Jaro(S_1,S_2)}} = = \frac{\mathbf{1}}{\mathbf{3}}\left(\frac{\mathbf{c}}{\mathbf{|S_1|}} + \frac{\mathbf{c}}{\mathbf{|S_2|}} + \frac{\mathbf{c - t/2}}{\mathbf{c}}\right)$$

Where **c** is the number of common characters and **t** is the number of transpositions, $|S_1|$ and $|S_2|$ is the length of the S_1 and S_2 respectively.

For Example Lets compare S_1='Taylor' with S_2 ='Tyalor'

Here c= 6, t=2, $|S_1|=|S_2|=6$ We got $Sim_{Jaro}(S_1,S_2)$ = 0.9444

Also Lets compare S_1='Taylor' with S_2 ='Sailor'

Here c= 4, t=0, $|S_1|=|S_2|=6$ We got $Sim_{Jaro}(S_1,S_2)$ = 0.7778

The above calculations means that 'Tyalor' have a better match to 'Taylor' with compare to 'Sailor' , and this result is identical with the human judgment .

Using of Jaro's algorithm the transposition of two characters causes less or equal of down weighting of similarity than substitution. For example compare 'Martha' with 'Marhta' and 'Jonathan' with 'Jonathon', the values of Jaro's similarity is 0.9444 and 0.9167 respectively. Table 2.3 shows examples of Jaro's Similarities for people names.

S_1	S_2	$Sim(S_1,S_2)$
SHACKLEFORD	SHACKELFORD	0.970
DUNNINGHAM	CUNNIGHAM	0.896
NICHLESON	NICHULSON	0.926
JONES	JOHNSON	0.790
MASSEY	MASSIE	0.889
ABROMS	ABRAMS	0.889
ITMAN	SMITH	0.000
JERALDINE	GERALDINE	0.926
MARHTA	MARTHA	0.944
MICHELLE	MICHAEL	0.869
JULIES	JULIUS	0.889

Table 2.3: Examples of Jaro's similarities for names

the Jaro algorithm requires $O(|S_1|*|S_2|)$ for two strings of length $|S_1|, |S_2|$, mainly due to Step 2 in the algorithm , which computes the "common characters" in the two strings.

2.2.1.3 Q-gram Distance

A Q-gram is a subsequence of q items from a given sequence. The items in question can be phonemes, syllables, letters, words or base pairs according to the application. A Q-gram of size 1 is referred to as a "unigram"; size 2 is a "bigram" (or, less commonly, a "digram"); size 3 is a "trigram"; and size 4 or more is simply called a "Q-gram". The use of bigrams or trigrams for compression of field value can be used to measure the closeness of match. Letter Q-grams, including trigrams, bigrams, and/or unigrams, have been used in a variety of ways in text recognition and spelling correction. (Innerhofer-Oberperfler, 2004)

The notion of Q-grams for given a string σ , its Q-grams are obtained by "sliding" a window of length q over the characters of σ. Since Q-grams at the beginning and the end of the string can have fewer than q characters from σ we introduce new Σ characters "#" and "%" not in σ, and conceptually extend the string σ by prefixing or padding it with q - 1 occurrences of "#" and suffixing it with q -1 occurrences of "%". Thus, each Q-gram contains exactly q characters, though some of these may not be from the alphabet Σ .

The intuition behind the use of Q-grams as a foundation for approximate string processing is that when two strings σ_1 and σ_2 are within a small edit distance of each other, they share a large number of Q-grams in common. (Ukkonen, 1992)

For example consider the Q-grams of length q=3 for string "john smith" are { (##j), (#jo), (joh), (ohn), (hn_), (n_s), (_sm), (smi), (mit), (ith), (th%), (h%%) }. Similarly, the Q-grams of length q=3 for "john a smith", which is at an edit distance of two from "john smith", are { (##j), (#jo), (joh), (ohn), (hn_), (n_a), (_a_), (a_s), (_sm), (smi),(mit), (ith), (th%), (h%%) }. If we ignore the position information, the two Q-gram sets have 11 Q-grams in common. Interestingly, only the first five Q-grams of the first string are also Q-grams of the second string.

The Q-grams similarity metric between two strings is constructed ranging from 0 to 1.0 using a normalized formula.

$$\mathbf{Sim}_{\mathbf{Q-grams}\ (S_1,S_2)} = \frac{1}{2}\left(\frac{|G_{S_1} \cap G_{S_2}|}{|G_{S_1}|} + \frac{|G_{S_1} \cap G_{S_2}|}{|G_{S_2}|}\right)$$

Where $|G_{s1} \cap G_{s2}|$ is the number of common Q-grams between S_1 & S_2, $|G_{s1}|$ and $|G_{s2}|$ is the number of Q-grams of s_1 and s_2 respectively.

For the above example we have $|G_{s1}|=12$, $|G_{s2}|=14$, $|G_{s1} \cap G_{s2}|=11$, so the similarity between them is 0.8511 according to the above formula.

Also consider a Q-grams of length q = 3 for S_1 = 'Street' can be constructed as Gs_1= {##S, #St, Str, tre, ree, eet, et#, t##} , and S_2 = 'Steret', are Gs_2 = {##S, #St, Ste, ter, ere, ret, et#, t##} . The two strings have 4 Q-grams in common. We have $|G_{s1}|=8$, $|G_{s2}|=8$, $|G_{s1} \cap G_{s2}|=4$, so the similarity between them is 0.5 according to the above formula.

Figure 2.5 shows the algorithm for the calculation of the similarity metric between two different strings S_1 and S_2. (Innerhofer-Oberperfler, 2004)

```
Algorithm Q-gram (S1, S2)
// Input: two strings S1 and S2
// Output: matching-quota between S1 and S2 in %
   match = 0, i = 0, j = 0
  Generate the list GS1 from S1 and the list GS2 from S2
  Sort GS1 and GS2
 While (i < |GS1 | and j < |GS2 |)
          if GS1 (i) = GS2 (j) then
                  Match + +
                   i + +
                   j + +
              Else
                   if Gs1 (i) < GS2 (j) then
                       i + +
                   else
                       j + +
                   endif
 endwhile
Return ((match / |Gs1 | + match / |Gs2 |) / 2)
End
```

Figure 2.5: Algorithm for calculating Q-grams similarity metric (Innerhofer-Oberperfler, 2004)

For the Q-grams complexity when sorting the lists Gs with the appropriate use of hash-based indexes, it takes O (n log n), with n = max ($|G_{S1}|$, $|G_{S2}|$). But the average time required for computing the Q-gram overlap between two strings O (max {$|S_1|$, $|S_2|$}). (Elmagarmid et al., 2007)

It's worth indicating that an extension to Q-grams is being adapted to add positional information (location of a Q-gram within a string) and to match only common Q-grams that are within a maximum distance from each other. For example, 'peter' contains the positional bigrams ('pe',0), ('et',1), ('te',2) and ('er',3). If a maximum distance of comparison is set to 1, then bigram ('et',1) will only be matched to bigrams in the second

string with positions 0 to 2. Positional Q-grams can be padded with start and end characters similar to non-positional Q-grams, and similarity measures can be calculated in the same three ways as with non-positional Q-grams. (Christen , 2006)

2.2.2 Token-based Similarity Metrics

Character-based similarity metrics work well for typographical errors. However, it is often the case that typographical conventions lead to rearrangement of words (e.g., "John Smith" versus "Smith, John"). In such cases, character-level metrics fail to capture the similarity of the entities. Token-based metrics try to compensate for this problem by focusing on the common terms shared by strings instead of characters. (Elmagarmid et al., 2007).

Token-Based metrics consider strings as consequences or sets of words as in S_1= {W_1 W_2 W_3 … W_n } and s2 = {W_1 W_2 W_3 … W_n } respectively , where each word as a sequence of characters as in character-based ring matching , this adds great flexibility for two obvious reasons :(Wang, 2008)

1- Stop words and punctuation, which have no significant in content representation, can be easily removed from strings and only meaningful strings are left to compare.
2- Abbreviations can be taken care by expressing words as sequences of characters as in St → Street and MEU → Middle East University.

One measure of the token-based metrics is the cosine similarity with the Vector Space Model. The basic idea is to represent strings using vectors in which the components are the term frequency-inverse document frequency (TF-IDF) weight of each token in strings, and then the similarity is computed as the product of these vectors. (Cohen et al., 2003). (Elmagarmid et al., 2007)

The simplest token–based field matching algorithm is the Jaccard Similarity metrics, which counts the number of common words N_c and the number of distinct words N_d of two strings in comparisons and take the ratio of N_c/N_d as the similarity degree of two strings. The simple field matching algorithms is very similar to the Jaccard metrics, in which the similarity degree is calculated by a simple formula

$$\mathbf{Sim}_{\mathbf{Jaccard}\ (S_1, S_2)} = \frac{2N_c}{|S_1| + |S_2|}$$

Where $|S_1|$ is the number of words in S_1 and $|S_2|$ is the number of words in S_2. (Wang , 2008)

2.2.3 Phonetic Similarity Metrics

While the above measures focus on the spelling aspect of strings, some strings may be phonetically similar but not similar in spelling such as “Smyth” and “Smis”. For this problem, several phonetic similarity measures are proposed based on phonetic codings such as Soundex, NYSIIS, ONCA, Metaphone, and Double Metaphone. These measures first convert strings into codes using a phonetic coding scheme based on their pronunciations, and then compare them by matching their phonetic codes. Only strings with the same phonetic code are considered to be duplicates. Most phonetic methods including all presented here have been developed mainly with English in mind. Several techniques have been adapted for other languages.

2.2.3.1 Soundex

Soundex algorithm began in a patent by Robert C. Russell in 1918; the Soundex search algorithm takes a written word, such as a person's surnames, as input and produces a character string that identifies a set of words that are phonetically alike. It is very handy for

searching large databases when the user has incomplete data. For example, the word “Kageonne” is phonetically similar to “Cajun” despite the fact that the string representations are very different. The phonetic similarity metrics are trying to address such issues and match such strings. (Elmagarmid et al., 2007)

The method used by Soundex is based on the six phonetic classifications of human speech sounds (bilabial, labiodentals, dental, alveolar, velar, and glottal), which in turn are based on where you put your lips and tongue to make the sounds. (idmatchsystems.com, 2010)

A basic algorithm for calculating Soundex coding of a string includes the following steps:

1. Capitalize all letters in the word and drop all punctuation marks. Pad the word with rightmost blanks as needed during each procedure step.
2. Retain the first letter of the word.
3. Change all occurrence of the following letters to '0' (zero): 'A', E', 'I', 'O', 'U', 'H', 'W', 'Y'.
4. Change letters from the following sets into the digit given:
    - 1 = 'B', 'F', 'P', 'V'
    - 2 = 'C', 'G', 'J', 'K', 'Q', 'S', 'X', 'Z'
    - 3 = 'D','T'
    - 4 = 'L'
    - 5 = 'M','N'
    - 6 = 'R'
5. Remove all pairs of digits which occur beside each other from the string that resulted after step (4).

6. Remove all zeros from the string that results from step 5.0 (placed there in step 3)
7. Pad the string that resulted from step (6) with trailing zeros and return only the first four positions, which will be of the form <uppercase letter> <digit> <digit> <digit>.

For example HERMAN" will code to "H06505" which will then reduce to "H650", Table 2.4 shows more examples of the Soundex coding.

Surnames	Letters Coded	Coding
Allricht	l, r, c	A462
Eberhard	b, r, r	E166
Engebrethson	n, g, b	E521
Hanselmann	n, s, l	H524
Henzelmann	n, z, l	H524
Lind, Van	n, d	L530
Lukaschowsky	k, s, s	L222
McDonnell	c, d, n	M235
McGee	C	M200
O'Brien	b, r, n	O165
Opnian	p, n, n	O155
Oppenheimer	p, n, m	O155
Swhgler	s, l, r	S460
Riedemanas	d, m, n	R355
Zita	T	Z300

Table 2.4: Examples of Soundex coding

Soundex acts as a bridge between the fuzzy and inexact process of human vocal interaction, and the concise true/false processes at the foundation of computer communication. As such, Soundex is an inherently unreliable interface. For this reason, Soundex is only usable in applications that can tolerate high false positives (when words that don't match the sound of the inquiry are returned) and high false negatives (when words that match the sound of the inquiry are not returned). This limitation is true even of the best Soundex improvement techniques available.

A major drawback of Soundex is that it keeps the first letter, thus any error or variation at the beginning of a name will result in a different Soundex code. (Christen , 2006)

Other limitations of Soundex that were designed only for the 26 letters of English and the sound mapping cover Anglo-Saxon names only or primarily the Caucasian surnames, but works well for names of many different origins (such as those appearing on the records of the US Immigration and Naturalization Service). However, when the names are Latin and Chinese, this code is less satisfactory because much of the discriminating power of these names resides in the vowel sounds, which the method ignores. (idmatchsystems.com, 2010)

2.2.3.2 Metaphone Coding Method

This method was used for matching words that sound alike and is based on commonplace rules of English pronunciation. Metaphone ignores vowels after the first letter and reduces the remaining alphabet to sixteen consonant sounds, although vowels are retained when they are the first letter. Duplicate letters are not added to the code. Zero is used to represent the 'th' sound since it resembles the Greek theta when it has a line through it, and 'X' is used for the 'sh' sound. The sixteen consonant sounds are: B X S K J T F H L M N P R Ø W Y. (Lait & Randell, 1998)

2.2.3.3 Phonex and Phonix

Phonex is a variation of Soundex that aims to improve the encoding quality by preprocessing names according to their English pronunciation. Phonix goes a step further than Phonex and applies more than one hundred transformation rules on groups of letters. Some of these rules are limited to the beginning of a name, some to the end, others to the middle, and some will be applied anywhere. (Christen , 2006)

2.2.3.4 New York State Identification and Intelligence System (NYSIIS)

The NYSIIS system differs from Soundex in that it retains information about the position of vowels in the encoded word by converting most vowels to the letter A. Furthermore, NYSIIS does not use numbers to replace letters; instead, it replaces consonants with other, phonetically similar letters, thus returning a purely alpha code (no numeric component). Usually, the NYSIIS code for a surname is based on a maximum of nine letters of the full alphabetical name, and the NYSIIS code itself is then limited to six characters. Taft compared Soundex with NYSIIS, and concluded that NYSIIS is 98.72 percent accurate, while Soundex is 95.99 percent accurate for locating surnames. (Elmagarmid et al., 2007)

2.3 Decision Models

So far we have described methods that can be used to match individual fields of a record. In most real-life situations, however, the records consist of multiple fields; making the duplicate detection problem much more complicated and a decision must be made as to whether consider these records as duplicate (match), not duplicate (unmatched) or possibly duplicate (possibly match) . (Wang, 2008)

Many methods may be used for matching records with multiple fields. The presented methods can be broadly divided into two categories: (Elmagarmid et al., 2007)

- Approaches that rely on training the data to "learn" how to match the records. This category includes (some) probabilistic approaches and supervised machine learning models.

- Approaches that rely on domain knowledge or on generic distance metrics to match records. This category includes approaches that use declarative languages for matching and approaches that devise distance metrics.

2.3.1 Probabilistic Matching Models

After standardizing the data sets and choosing the similarity measures, the next step is to match every pair of potential duplicates and resolve the duplicates. To compare records, the duplicate detection process is modeled as a probabilistic model in which the probability is the normalized similarity score of two records between 0 and 1. Depending on the similarity score and the given threshold, pairs of records are labeled as match, possible match or unmatched. Monge and Elkan propose several string matching algorithms for detecting duplicated records in the database. They first introduce a basic field matching algorithm which treats fields as a sequence of atomic strings (sorted) and computes the matching score by counting the number of matched strings. To address the problem of abbreviations and ordering of strings, they propose a recursive field matching algorithm which compares every pair of strings and chooses the maximum score as the similarity score of two strings. Cohen implements the WHIRL system that uses the cosine similarity measure, together with the vector space model, to measure the similarity of records. (Elmagarmid et al., 2007)

Fellegi and Sunter proposed in 1969 that a probability that a field agrees given the record pair examined is a matched pair (m probability) ,its also uses the probability that a fields agrees given the records pair examined is an unmatched field (u probability) . The value of u is calculated by occurrences of field entries within a database. Then u needs to be

recalculated after entry of new data or segment of new data. Guessing what the occurrences of error is in various field approximate the value of m. The weight of matching field is computed as $\log_2$ (m/u) and the weight of field disagreement is $\log_2$ (1-m)/ (1-u). The composite weight of record matching is the sum of the weight for individual fields. The greater the composite weight, the greater the probability the records are the same. (Wang, 2008)

Chaudhuri propose a fuzzy similarity measure to match records in databases. They define three transformation operations: token replacement, token insertion, and token deletion, each of which is associated with a cost. The similarity of two records is then measured by computing the cost required to transform one tuple into the other using these operations. Note that this method is different from the edit distance in that the fuzzy matching function focuses on token level operations while the edit distance operates at the character level. (Chaudhuri et al., 2003)

2.3.2 Supervised-Learning-Based Approaches

A problem with Probabilistic and rule-based approaches is that the weights of different attributes and the threshold for optimal resolution results are difficult to decide. Also, some types of records may require different measures for different attributes. Manually choosing the measures or weights for each attribute is not only tedious but subject to produce errors. In the presence of some labeled samples, a number of supervised learning approaches that incorporate the properties of data sets into the similarity measures are proposed for duplicate detection. Therefore supervised learning systems rely on the existence of training data in the form of record pairs, prelabeled as matching or not. (Elmagarmid et al., 2007)

Bilenko compare several similarity measures on different benchmark data sets, they found out that no single similarity measure can perform the best in all cases. They propose to use similarity vectors of two references to train a binary support vector machine (SVM) classifier and use the classifier's confidence in the match class as a new similarity measure for references. (Bilenko & Mooney, 2003)

2.3.3 Active-Learning-Based Approaches

One problem with supervised learning approaches is that a large training data set is required to train a classifier and preparing such a training data set that contains enough negative or positive representative cases is difficult. To reduce the size of training data sets and still achieve high accuracy, some methods have been proposed using active learning approaches.

An active learner starts with a small labeled training data set and a large pool of unlabeled data items. It then actively picks out the important records that when labeled will improve the performance of the learner during the learning process. Each time new labeled records are added into the training data set, the learner will be retrained. ALIAS system is a learning-based duplicate detection system which aims to minimize the size of labeled data sets. For ambiguous instances that are difficult to resolve, they introduce an uncertainty score for each instance and propose a classifier independent way to measure the uncertainty score of instances. Using the uncertainty score, the system picks out the most uncertain instances to be manually labeled and retrains the learner based on the users' feedback. Tejada uses a similar strategy for learning mapping rules. A mapping-rule learner in their

system actively chooses the most informative candidate mappings and thus reduces the number of training examples. (Elmagarmid et al., 2007)

2.3.4 Rule-Based Approaches

The probabilistic model enforces pre-defined rules on records. For example, a rule can be defined like this: if the similarity of two records is greater than threshold $ø_1$, then they are considered to be duplicates otherwise if the similarity is less than threshold $ø_2$, then they are not duplicates. The rule-based approaches aim to categorize records using these pre-defined rules or rules learned from training data sets.

Hern´andez and Stolfo suggest the use of an equational theory to model the logic of domain equivalence. In their system, the rules for their test data are first described and evaluated using a declarative rule language and then converted into a more efficient C implementation. The logic of these rules is expressed using well chosen similarity measures and thresholds like those introduced above. For example, if two people have similar name spellings and these people have the same address, we may infer that they are the same person. We can note here that the selection of distance function and a proper threshold is knowledge intensive activities that demand experimental evaluation. Figure 2.6 shows an example of rules defined in their method. (Hernández & Stolfo, 1998)

Given two records, r1 and r2.
IF the last name of r1 equals the last name of r2 ,
AND the first names differ slightly ,
AND the address of r1 equals the address of r2
THEN
r1 is equivalent to r2.

Figure 2.6: Simplified rule in English to illustrate the equational theory (Hernández & Stolfo, 1998)

Shahri uses fuzzy logic and machine learning approaches to define the degree of match between a pair of records by mapping of similarity values to linguistic concepts such as "matched", "possible matched" and "unmatched" using two step process by first determining a quantitative measure of the comparison or attribute in question and assign a number between 0 and 1 to represent how strongly the comparison or attributes measures relates to the linguistic concepts. Then devising fuzzy rules to be used in the inference engine implemented in an expert system, which uses attribute similarities for comparing and detecting the duplicates. The advantages of utilizing fuzzy logic for fuzzy duplicate elimination include the ability of specifying the rules in natural language easily and intuitively so removing the hard-coding process. (Shahri & Shahri, 2006)

2.4 Duplicate Detection Tools

Next we highlight some of the software tools being developed by the record linkage and duplicate record detection research community as an open architecture for academic and commercial use. The importance of these software tools came from its rich architecture that can help us in development of such tools.

2.4.1 FEBRL

FEBRL is an open source data cleaning software that provides data standardization and probabilistic record linkage with choices of methods for blocking and comparison functions. FEBRL's data standardization primarily employs a "supervised machine learning approach" implemented through a novel application of hidden Markov models. For duplicate detection, FEBRL implements a variety of string similarity metrics, such as Jaro, edit distance, and Q-gram distance. FEBRL also supports phonetic encoding (Soundex,

NYSIIS, and Double Metaphone) to detect similar names. Since phonetic similarity is sensitive to errors in the first letter of a name, FEBRL also computes phonetic similarity using the reversed version of the name string, sidestepping the "first-letter" sensitivity problem. Figure 2.7 shows processing view of standard record linkage system architecture as implemented in FEBRL. (Gu et al., 2003) (Baxter et al., 2003) (Elmagarmid et al., 2007)

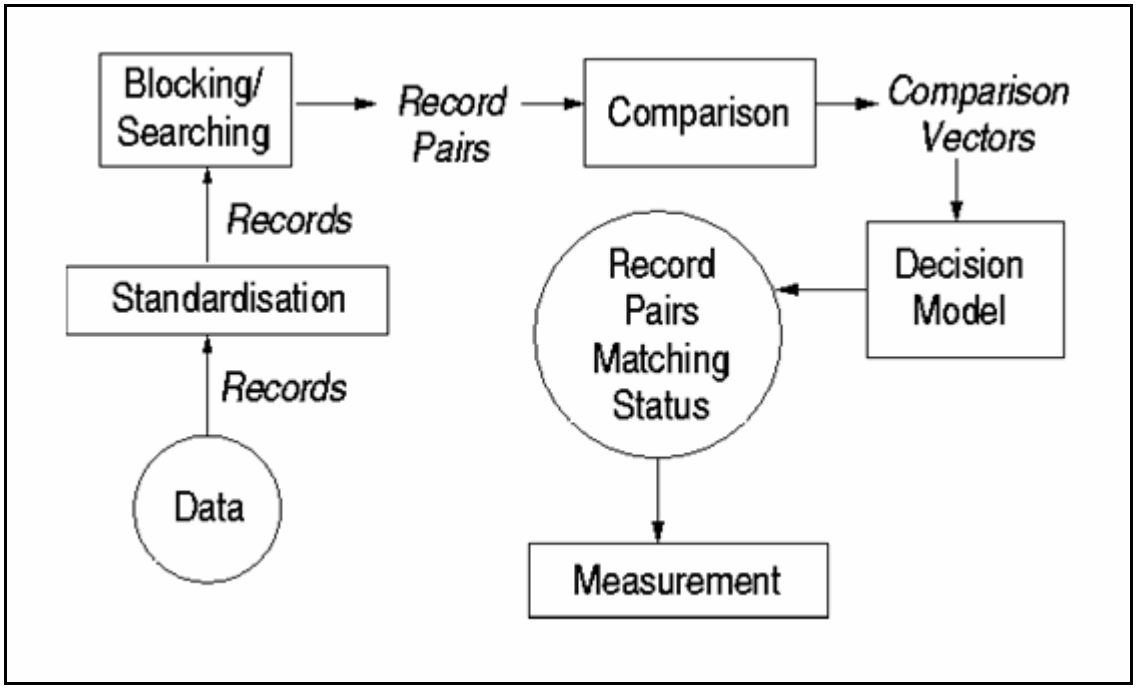

Figure 2.7: Processing view architecture in FEBRL. (Baxter et al., 2003)

2.4.2 FRIL

FRIL is a fine-grained record integration and linkage tool (FRIL) that extends traditional record linkage tools with a richer set of parameters. The users may systematically and iteratively explore the optimal combination of parameter values to enhance linking performance and accuracy. FRIL implements several search or blocking methods such as the sorted neighborhood method.

Results of linking birth defects monitoring program and birth certificate data using FRIL show 99% precision and 95% recall rates when compared to results obtained through handcrafted algorithms, and the process took significantly less time to complete. Experience and experimental result suggest that FRIL has the potential to increase the accuracy of data linkage across all studies involving record linkage. In particular, FRIL will enable researchers to assess objectively the quality of linked data. Figure 2.8 shows The FRIL architecture. (Jurczyk et al., 2008)

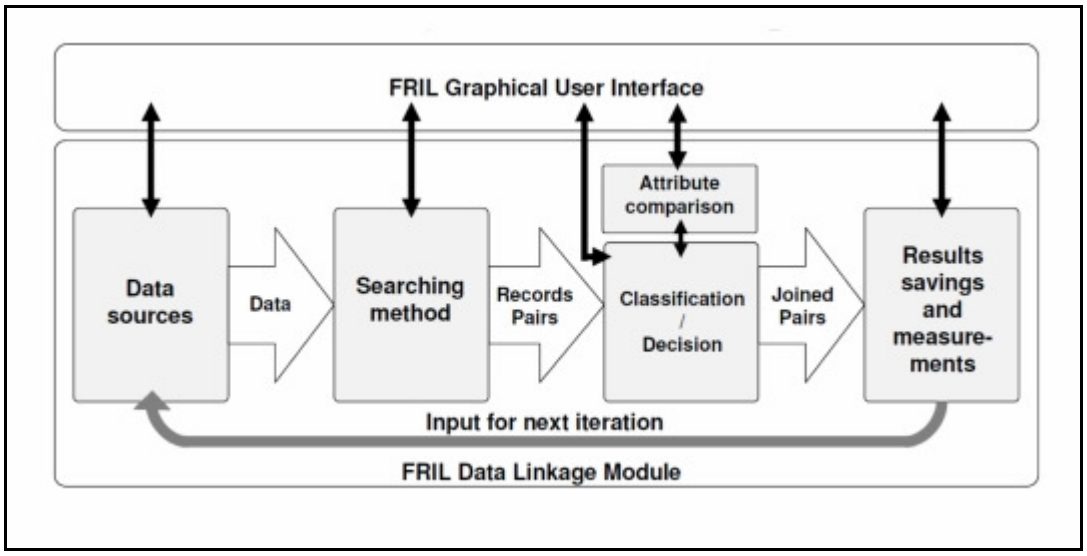

Figure 2.8: FRIL architecture. (Jurczyk et al., 2008)

2.4.3 TAILOR

TAILOR is a record linkage toolbox that follows a layered design, separating comparison functions from the duplicate detection logic, and can be used to build a complete record linkage and duplicate record detection models by tuning a few parameters and plugging in some in-house developed and public domain tools that includes performance and accuracy metrics to compare these different models. TAILOR incorporates other ready-made tools

in order to provide additional functionality such as MLC++ that contains, among other things, classification methods that are used by TAILOR in both the induction and the hybrid record linkage models. Another tool is called DBGen which is used to generate synthetic data files. The operation of DBGen is controlled by a large number of parameters such as data size, duplication rate, and error probabilities in the various fields. Figure 2.9 shows TAILOR Information Flow Diagram (Elfekey et al., 2002)

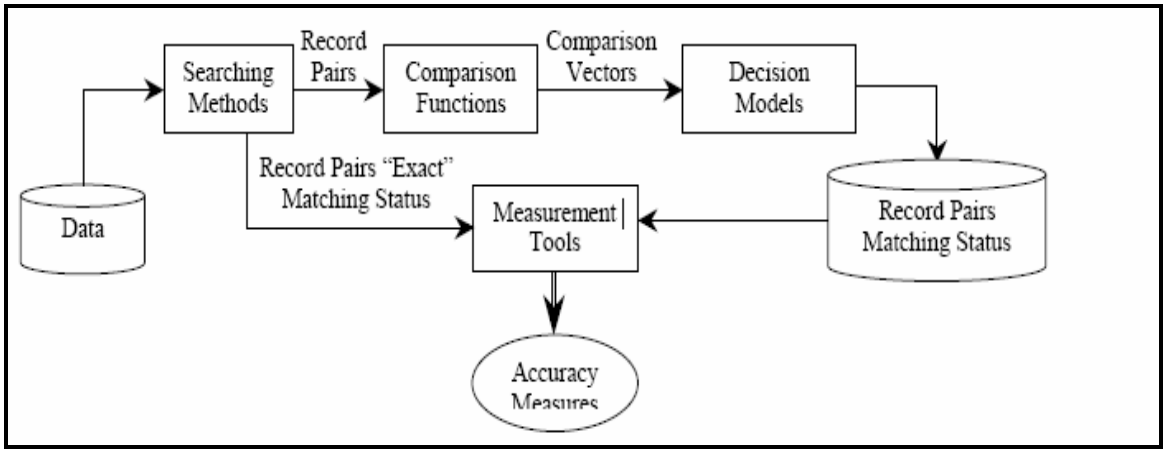

Figure 2.9: TAILOR *information* flow diagram. (Elfekey et al., 2002)

2.4.4 BigMatch

BigMatch is a record linkage and duplicate detection program tool used by the US Census Bureau that allows to run several different blocking criteria for two relations (a large file and a moderate size file) by just requiring the large file to be read just once and without requiring prior sorting of either file. The only requirement is that one of the two relations should fit in memory. The required sorting is done on a key list in memory, and the time required to run the program is generally modest. For each blocking criterion, the program outputs a file of records from the large file that are plausible matches to records in the moderate file. (Yancey, 2007)

Most standard US Census Bureau record linkage program features one-to-one matching which results in each record being paired with its most likely match within its blocking group. BigMatch program does not do this, so that an output file may contain several records from the large file that were stored as likely matches to the same record in the moderate file. The purpose of the BigMatch program is to function as a preprocessor that can efficiently extract smaller files from a very large file so that these smaller files can be used efficiently with standard record linkage software. However, the BigMatch program has also been used for deduplicating a single file. If a file contains multiple duplicates, then the absence of one-to-one matching can be advantageous, provided that the file can fit in core memory. (Elmagarmid et al., 2007)

2.5 Related Studies

There has been a lot of work and researches in the field of duplicate record detection and many systems have been implemented with different approaches to blocking. In this section we are going to introduce some of the most important related studies in this field which provides us a good guidance to our work.

- (Hernández & Stolfo, 1998): The authors introduced a system for accomplishing data cleansing tasks and demonstrate its use for cleansing lists of names of potential customers in a direct marketing-type application. The system provides a rule programming module using an intelligent "equational theory "in addition to the sorted neighborhood method as a blocking method.

- (Gu et al., 2003): The authors presented the current standard practices in record linkage methodology. The definition of the record linkage problem, the formal probabilistic model and an outline of the standard practice algorithms and its recent proposals. And concludes with a summary of the current methods and the most worthwhile research directions.
- (Christen 2007): The author evaluated the traditional, as well as several recently developed, blocking methods within a common framework with regard to the quality of the candidate record pairs generated by them. Also propose modifications to existing blocking methods that replace the traditional global thresholds with nearest-neighbor based parameters.
- (Elmagarmid et al., 2007): The authors presented a thorough analysis of the literature on duplicate record detection covering similarity metrics that are commonly used to detect similar field entries. They also present an extensive set of duplicate detection techniques that can detect approximately duplicate records in a database. They also cover multiple methods for improving the efficiency and scalability of approximate duplicate detection algorithms. In addition to coverage of existing software tools with a brief discussion of the big open problems in this filed.
- (Draisbach & Naumann, 2009): The authors in this paper briefly introduced and analyzed two popular families of methods for duplicate detection. Blocking methods and Windowing methods. Also the authors proposed a generalized

algorithm, the Sorted Blocks method and compared the approaches qualitatively and experimentally.

- (Devries et al., 2009): The authors introduced an improvement to the suffix array blocking method, together with an in-depth analysis and experimental results showing the effectiveness of the method on real and synthetic data. Also compare the improved method against the base method of suffix array blocking as well as the well-known traditional blocking methods.
- (Tamilselvi & Saravanan, 2009): The authors present a general sequential framework for duplicate detection and elimination using a rule-based approach to identify exact and inexact duplicates and to eliminate duplicates. The proposed framework uses six steps to improve the process of duplicate detection and elimination. Also the authors compare this new framework with previous approaches using the token concept in order to speed up the data cleaning process and reduce its complexity. Analysis of several blocking key is made to select best blocking key to bring similar records together through extensive experiments to avoid comparing all pairs of records.

CHAPTER 3

DUPLICATE DETECTION FRAMEWORK DESIGN

Our design for duplicate record detection framework takes its roots from the generic frameworks suggested and used in solving record linkage and duplicate detection problems. (Gu et al., 2003), (Tamilselvi & Saravanan, 2009)

In Figure 3.1 we show a sequential based framework developed for detection of duplicate records system. Each stage in this framework represents a part or subsystem of the whole system. In our implementation for evaluation of blocking methods, we have only implemented the needed stages or part of them required for basic functionality to help us focus on problem at hand. The stages on this framework are as follow:

A. **Data preparation**: This stage refers to the process in which our data set would be processed and stored in a uniform manner in the database.
B. **User Interaction**: In this stage the user would interact with the system through the user interface for the selection of cleaned standardized data set, selection of attributes to be used as a key and supply the parameters required for the blocking methods.
C. **Blocking key value generation**: In this stage the blocking key is being generated from attributes of each record and data is being sorted.
D. Blocking: In this stage in order to reduce our comparison space, a blocking method is being used to group the data into blocks.

E. **Similarity computation & comparison:** By using of similarity metrics repository the similarities is being calculated and compared for each pair of records.

F. **Decision model:** In this stage we use a decision model to identify records as duplicates or not duplicates and record this information in our log files.

Next we discuss each stage in detail.

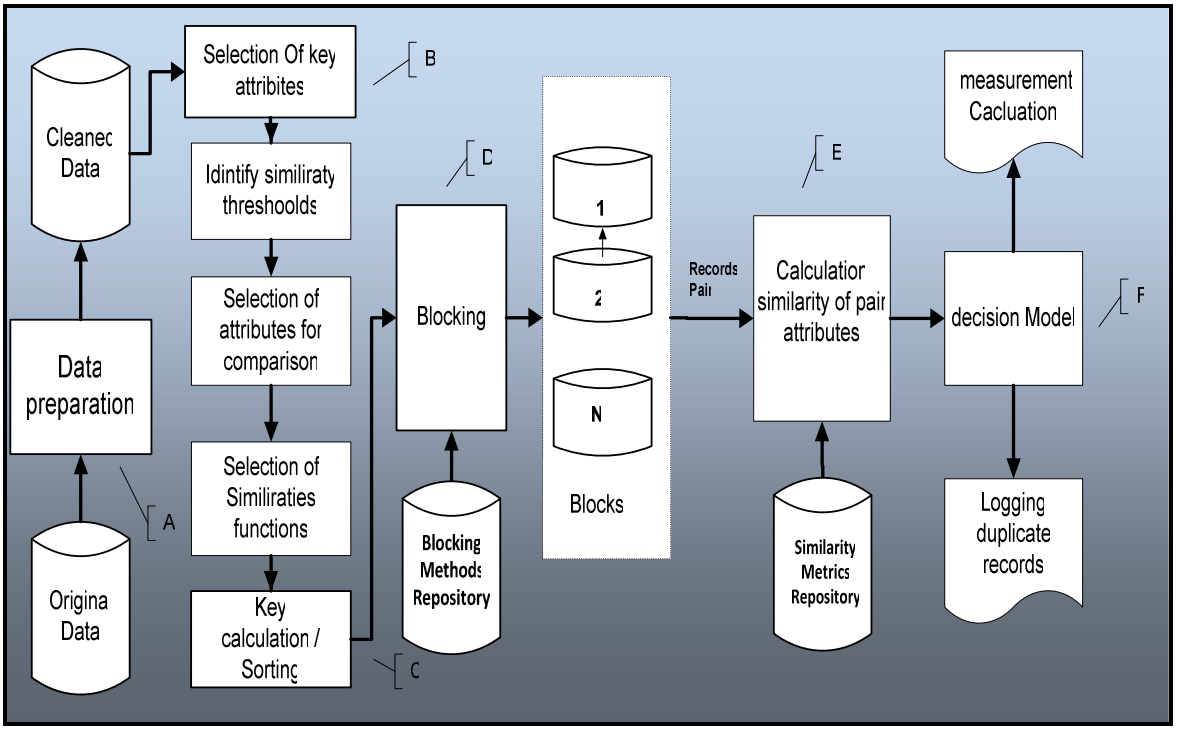

Figure 3.1: Framework for duplicate detection

A: Data preparation:

When integrating data from different sources, many issues may cause the irregularity problem in the data set. Data sets from different sources are usually collected by different sources using different means and tools. The format used in one data set may not be the format used in other data sets, and these data sources may follow different conventions. So a data preparation stage is needed during which data entries are stored in a uniform manner

in the database, resolving the structural heterogeneity problem. This stage is an optional stage depending on the quality of the data, but mainly the data preparation stage includes parsing, data transformation and standardization steps. (Elmagarmid et al., 2007)

1- Parsing: In which we locates, identifies and isolates individual data elements in the source files. Parsing makes it easier to correct, standardize, and match data because it allows the comparison of individual components, rather than of long complex strings of data.
2- Data transformation: Refers to simple conversions that can be applied to the data in order for them to conform to the data types of their corresponding domains.
3- Standardization: Without standardization, many true matches could be wrongly identified as unmatched because the common identifying attributes do not have sufficient similarity. The basic functionality of this step is to

    a. To replace many spelling variations of commonly occurring words with standard spelling.
    b. To standardize the representation of various attributes, to the same system of units, or to the same coding system. For example, 0/1 instead of M/F for a 'gender' attributes.
    c. To perform integrity checks on attribute values or combinations of attribute values.

B: User Interaction

After the data preparation stage, the user interacts with system through the user interface to facilitate doing several functionalities such as:

- Identifying and selection of the required data set.
- Identifying and selection of attributes used in building of blocking key value.
- Entry of parameters needed for different blocking methods such as the window size, overlapping block size, minimum suffix length, and maximum block size.
- Identifying and selection of attributes used in comparison.
- Identifying of similarities functions used for comparing pair attributes such as edit distance, Q-gram, etc.
- Identifying of similarity threshold values used in decision model.
    - Facilitate the tracing of results at each step to identify and evaluate any possible illogical or wrong results, also support for the presentation and displaying of the results of quality and complexity measurements for blocking methods such as reduction ratio, pair completeness and processing time.

C: Blocking key value generation

After selection the required key attributes, for each record we create the key by extracting relevant fields or portion of the field's. We have different choices here:

1- A blocking key taken from a single record attribute or its parts values, like the value of surname attribute or the first two initial letters for given mane attribute only.

2- By concatenation of values (or parts of them) from several attributes. For example concatenation of a person surname with the first 3 characters of a person given name for each record.

3- Using phonetic encodings, for example in our experiments we used a key value composed of the Soundex code of surnames (4 characters) concatenated by the first four non vowels (constant) characters of given names and the first 4 digits of social security numbers.

In this stage we mainly depend on our Database Management System for some of the background activities needed such as navigating, retrieving of related records during blocking key value generation , also provides us with some helpful built-in tools such as sorting and indexing functionalities that is required after the key generation .

D: Blocking

The blocking stage is used to reduce the huge number of comparisons of record pairs by bringing potentially duplicate record pairs together. Using a blocking method from the blocking methods repository we can generate blocks of records where each pair is being compared. In this stage several blocking methods such as the standard blocking, sorted neighborhood, or suffix array could be used. Several parameters supplied in the user interaction stage are needed such as the minimum suffix length and maximum block size for blocks generation.

E: Similarity computation & comparison

Using similarity metrics repository the records pair are being compared with each others and for each pair of records the similarities is being calculated. We can compare each

attribute with the corresponding one or select attributes that have sufficient information's content to support the detection process and got the similarity weight. (Gu et al., 2003)

The attribute selection step is the foundation step for all the remaining steps and very important to reduce the time and effort for the further work such as elimination process.

For example, a field such as gender only has two value states and therefore could not convey enough information to identify a match uniquely. On the other hand, a field such as surname conveys much more information, but it may frequently be recorded incorrectly. In our implementation we have chosen given names, surnames, addresses as the most important attributes that characterize our data set.

Duplicate detection processes depend on string similarity functions for record fields and similarity computation functions depend on the data type. Therefore the user must choose the function according to the attribute's data type, for example numerical, string and so on.

Several string similarity measures or domain specific measures to compute the attribute similarity can be employed here such as Character-level or Token-based metrics.

F: Decision Model

Generally in this stage the records can be identified as duplicates or not duplicates, and this information is being recorded in the log file and numbers is being collected for measuring the quality and performance of the blocking technique.

Once matching weights of individual attributes of two records are calculated, the next step is to combine them to form a composite weight or score and then decide whether a record pairs should be matched or unmatched.

The simplest way to calculate the composite weight is to use the average of all the matching weights assuming each attribute contributes equally. If some knowledge on the importance of individual attributes is available, the weighted (fixed) average can be used.

In our implementation we have followed the equational theory proposed by Hern´andez and Stolfo , where we have inferred simple rules that dictate the logic of the data set domain by using a proper threshold to detect whether two records have a match or don't match. For example if two persons have a high degree of similarity in their given name and surnames and have an intermediate similarity in their addresses then the two records are considered matched. We have chosen a 0.8 for high similarity and 0.6 for an intermediate, these numbers are very important to the accuracy and efficiency, and have been chosen after several runs to ensure that we have high precession in identifying that two records are a possible candidate match and at the same time be able to detect a large number of matching records. During this stage numbers could be collected such as the total number of records in the data set, total number of identified possible duplicates, total number of true duplicates, total number of false positives (non-duplicates which where classified as duplicates), total number of comparisons performed and processing time.

Using the numbers collected in this stage the quality, accuracy, efficiency and complexity of the blocking methods are being calculated using the above numbers. The results after each run would be like reduction ratio, pair's completeness, pair's quality, F-score.

CHAPTER 4

EXPERIMENTAL EVALUATION

In this chapter, we present the conducted experiments with their used settings and their results. The experiments are conducted on a synthetic data set that was previously used as benchmark in other related work. The data set is presented in detail in Section 4.1.The development environment is presented in Section 4.2. In Section 4.3, we introduce the performance measures used for evaluating and comparing the quality and complexity of blocking methods.

The experiments in Section 4.4 evaluate the performance of two recently developed, blocking methods, the sorted blocks and standard suffix array and its improvement, with two older methods, the standard blocking and sorted neighborhood blocking, using standard measures of accuracy, efficiency and quality.

4.1 Data Set

The data set we have used for the evaluation is called FEBRL (Freely extensible biomedical record linkage) data set. The FEBRL data set contains patients data such as names (given and surname), addresses, ages, phone numbers and social security numbers. This data set contains 9968 records among them there are 6000 original records and 3968 duplicated records. There is up to 9 duplicates for an original record, a maximum of 3 modifications per attribute, and a maximum of 10 modifications per record in each duplicated record. Table 4.1 shows a sample duplicate records from this data set.

GIVEN_NAME	SURNAME	ADDRESS	AGE	PHONE_NUMBER	SOC_SEC_ID
Hannah	urquhart	florina place	-	03 50770094	1614610
hann ah	urquzart	florihaplace	-	03 50770094	1614510
Teagan	Upton	Wilkins street	18	07 45868407	1198233
Teagan	Uptiny	Wilkins street	18	07 45867407	1198233
Teatan	Upton	Wilkins street	-	07 45886407	1198233
Brooke	Uren	barrett street	31	03 38165026	5411400
Broole	Umrh	barrettaareet	31	03 38165026	5421400
Brooke	Urpn	barrettistreet	31	03 38170526	5411050

Table 4.1 Sample duplicate records from the FEBRL data set (datamining.anu.edu.au)

4.2 Environment

To evaluate and compare the various blocking methods, all the methods have been implemented using Java and Oracle Database 10g Express Edition (Oracle Database XE) environment. The development environment used was Netbeans IDE for compiling and executions. The experiments were carried out on a Dell server equipped with an Intel Xeon Quad Core of 2.66 GHz and 2GB RAM, running MS Windows 2003 standard server operating system.

4.3 Performance Evaluation Measures

The performance of blocking methods have traditionally been evaluated using the *reduction ratio*, *pairs completeness*, *F-score*, and *pairs quality* measures (Christen & Goiser, 2007), as defined below.

Let N_M and N_U be the total number of matched and un-matched record pairs, respectively, such that $N_M + N_U = |A| \times |B|$ for linkage of two data sets A and B, and $N_M + N_U = |A| \times (|A|-1)/2$ for deduplication of a data set A. Next, let S_M and S_U be the number of true matched and true unmatched record pairs generated by a blocking method, respectively, with $(S_M + S_U) << (N_M + N_U)$.

Reduction Ratio

The measure of efficiency is typically carried out with the use of the Reduction Ratio measure (RR), which quantifies the reduction the number of record pairs to be compared in detail in the candidate set of the comparison space, i.e. where the more record pairs are removed by a blocking method the higher the reduction ratio value becomes (Christen, 2007).

The RR is simply the ratio of the number of records ($S_M + S_U$) in the candidate set as decided by the blocking method, and the total number of records that would be in the candidate set and matched against when no blocking method is used ($N_M + N_U$). The result is then taken as the difference from 1, to cause a higher RR to equal a more desirable result. The RR measures the relative reduction of the comparison space, but without taking into account the quality of the reduction, i.e. how many record pairs from U and how many from M are removed by the blocking process. The RR measure is given by:

$$RR = 1 - \frac{S_M + S_U}{N_M + N_U}$$

Pairs Completeness

Accuracy measurement for blocking methods is usually carried out with the use of the Pairs Completeness measure (PC). This measure is simply the ratio of the number of true matches (S_M) generated by the blocking algorithm correctly includes in the candidate set to be matched, and the total number (N_M) of true matches that exist in the data set and would all be found when using no blocking. It measures how effective a blocking method is in generating true matched record pairs. Pair's completeness therefore corresponds to the

recall measure used in information retrieval and it measures the coverage of true positives (De vries et al., 2009). The PC is given by:

$$PC = \frac{S_M}{N_M}$$

F-score

Both PC and RR measures should be maximized, but there is a tradeoff between them. F-score measure captures this tradeoff by combining PC and RR via a harmonic mean. So it's a measure of accuracy of the experiment (Yan et al., 2007). The F-score is given by:

$$F-score = 2 * \frac{RR * PC}{RR + PC}$$

Pairs Quality

The measure of the quality of the blocking method is typically carried out with the use of the pairs quality (PQ). It is the number of true matched record pairs generated by a blocking method (S_M) divided by the total number of record pairs returned ($S_M + S_U$). A high pair's quality means a blocking method is efficient and mainly generates true matched record pairs, while a low PQ means a large number of true unmatched are also generated. This results in more record pair comparisons that have to be made, which is computationally expensive. The PQ corresponds to the precision measure as used in information retrieval (Christen, 2007) (Christen & Goiser, 2007). The PQ is given by:

$$PQ = \frac{S_M}{S_M + S_U}$$

4.4 Experiments

We have conducted two sets of experiments, where in the first one we have compared both windowing and blocking methods with sorted blocks method and in the second one we have compared the improved suffix array method against both standard suffix array method and standard blocking method.

4.4.1 Experiment 1

Our first set of experiments is designed to compare the four blocking methods, using the standard performance measures defined in Section 4.3. The methods are:

1. Standard blocking.
2. Sorted neighborhood method.
3. Sorted blocks method (overlapping blocks are quarter of the block size)
4. Sorted blocks method (overlapping blocks are half of the block size)

For each method we have used a window or block size that range from 2 up to 200 records each and all of the four methods have been tested under the same conditions such as the sorting key, similarity functions and decision model.

For the blocking key value we used a key composed of the Soundex code of surnames (4 characters) concatenated by the first four non vowels (constant) characters of given names and the first 4 digits of social security numbers, in the case of missing or null attribute in a record or the number of characters less than the required we used a special characters padding at the end of each part. After key generation all the records are being sorted using the built in features of Oracle DBMS since it's highly optimized for performance.

The similarity functions used for comparison were Levenshtein edit distance, Jaro distance and Q-grams. We choose Levenshtein and Jaro distance for use since it can give us a good result in detecting similarity for surnames and given names especially for typographical and spelling errors, we used the Q-grams to handle spelling errors in addresses composed of more than one string or token , all these functions can handle insertion, deletion , substitutions in strings for matching of two attributes and correspondingly returned a value between zero and one , where zero means that there is no match and one full match.

For comparison of records to detect duplication we have inferred simple rules that are based on equational theory that dictate the logic of the data set domain, we used the following four rules to determine whether two records have a match or don't match:

1. If two persons have a high degree of similarity in their given names and surnames and have an intermediate similarity in their addresses.
2. If two persons have a high degree of similarity in their surnames and have the same social security numbers.
3. If two persons have a high degree of similarity in their surnames and have some intermediate similarity in their addresses.
4. If two persons have an intermediate similarity in surnames, given names, addresses using multiple edit distance functions.

The above rules are used in stages for example if the first rule wasn't satisfied we go to the next rule as so on.

The implementation of "having a high degree of similarity "and "intermediate degree of similarity "is based on computing of distance function for each related pair attributes in

records and comparing the result with a distance threshold, we have chosen a 0.8 for high similarity and 0.6 for an intermediate, these numbers are very important to the accuracy and quality , and have been chosen after several runs to ensure that we have high quality in identifying that two records are possible candidates and at the same time be able to detect a large number of matching records. Table 4.2 and Figure 4.1 show the effect of varying the value for threshold and its effect in the measurements percentages. We can see that a 0.8 and 0.9 is a good candidate for a threshold value. PQ is higher for 0.9 compare to 0.8 and PC is higher for 0.8 compare to 0.9 threshold values. So we chose 0.8 to detect a large number of true matches without compromising the quality at the same time.

Threshold	**Total Identified**	**True**	**False**	**PQ**	**PC**	**RR**	**F-score**
0.9	3171	1712	1459	**53.99**	**43.15**	99.99	60.28
0.8	4518	2067	2451	**45.75**	**52.09**	99.99	68.5
0.7	5458	2162	3296	39.61	54.49	99.99	70.54
0.6	8644	2205	6439	25.51	55.57	99.98	71.44
0.5	11388	2217	9171	19.47	55.87	99.98	71.68
0.4	15456	2229	13227	14.42	56.17	99.97	71.93

Table 4.2: Sorted neighborhood method with window size =10 and varying threshold

We should note that a similarity threshold of 0.78 used in other works for delivering good results for both precision (PQ) and recall (PC). (Draisbach & Naumann, 2009)

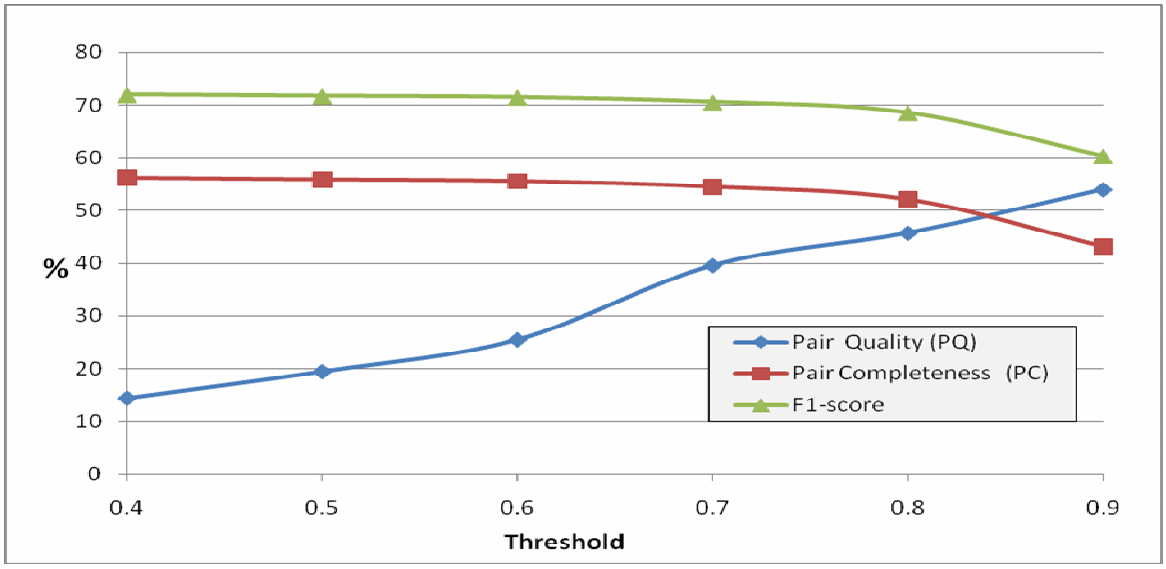

Figure 4.1: Sorted neighborhood method with window size =10 and varying threshold

We should note that the above chosen equational theory rules are used for experimental purposes and in real life such rules should be extended and built more accurately with the help of an expert from the problem domain.

Once a rule declares a possible match between any two records the result is being logged in the log table with additional information collected for measuring the quality and performance of the blocking method for statistical purposes.

4.4.1.1 Results Analysis

Comparing the results for quality we used the PQ measurement we have noted that the quality is decreasing whenever we increase the window or block size and this primarily due to the increasing number of true unmatched records detected by the blocking methods and the increasing number of records being compared. The standard blocking method has outperformed all other methods in PQ and quality is increasing whenever generalizing the standard blocking and increasing the number of overlapping blocks.

Comparing the results for accuracy we used the PC and have noted that the proportion of identified true duplicates is increasing in an accelerated way whenever block size is less than 20 and then it tends to increase in a linear fashion after this. Sorted neighborhood method outperforms all other methods in PC percentage and we were able to reach a percentage of 50% with a window size of 8 , standard blocking got the lowest percentage due to the fact that a lesser number of records is compared will decrease the chance of detecting duplicates. We have reached a maximum of 69% PC in best situations and this number can be increased with the help of optimizing our rule-based decision model and increasing number of detection rules although this may slightly decrease our precession.

The experiment also confirmed that the generalized algorithm of sorted blocks where overlapping block size is equal to block size -1 can outperform the sorted neighborhood method slightly especially in terms of PC but in the expense of processing time needed as shown in Table 4.3.

Window/block size	**Sorted Neighborhood Method**		**Sorted Blocks Method (overlapping =block size -1)**	
	PC	**Processing time (sec)**	**PC**	**Processing time (sec)**
2	25.3	2	25.5	10
8	52.7	9	54.6	14
14	54.4	11	56.2	23
26	58.6	21	60.4	31
38	61.5	30	63.7	50
50	63.4	38	65.5	63
65	64.8	49	66.7	79
77	65.7	58	67.5	93

Table 4.3: Sorted neighborhood method compared with Sorted blocks method where overlapping portion is equal block size -1

For RR which measures the reduction of the comparison space or how many records being removed by the blocking method all methods give almost similar results between 99.9886%

up to 99.998%, where standard blocking always outperform other methods in RR and this can be referred to the fact that the disjoint blocks decrease the chance for additional comparisons for unmatched records with the help of a key used for sorting the records. Sorted neighborhood method got the lowest RR percentage where our sliding moving window increases the number of non matching records, for sorted blocks methods the RR decreases whenever the overlapping size is increasing for a windows/block size of 50 and an overlapping partition half the block size of the precession is the same.

The F-score which calculate the mean of RR and PC and measure the accuracy of our test is showing similar results for all methods, where the F-score is increasing whenever moving from standard blocking through overlapping to the standard neighborhood method, after a windows size of 25 there is a slight chance of getting better results.

A note for the sorted blocks method where results show that standard blocking can be enhanced by using overlapping. We can see from the figures in Appendix 2 that blocking can be enhanced by using overlapping but its participation in detection is decreases whenever block size increases probably to the fact that larger disjoint blocks tend to have high degree of similar records.

In addition to the accuracy of blocking methods, we also measure the time efficiency for these methods, processing time collected during the tests shows clearly that sorted neighborhood method is having the highest processing time among all other methods, this can be reasoned by the high number of comparison performed and that number is always high with comparison to other methods.

Experiment 1 results graphs are shown in appendix 2.

4.4.2 Experiment 2

Our second set of experiments designed to compare the following blocking methods:

1. Standard suffix array method
2. Improved suffix array method using edit distance.
3. Improved suffix array method using Q-gram.
4. Standard blocking method.

Using the standard performance measures defined in Section 4.3.

The blocking key value (BKV) used in the above blocking methods based on using a concatenation of a person surname with the first 3 characters of a person given name for each record .

For standard blocking we used only the blocking values to group and compare records in each block that have the same BKV. For the remaining suffix array methods we build a suffix array inverted index that has 46906 unique values for the above data set. During processing we have used the minimum suffix length value of 2; also we have ignored all suffix blocks that have more than 50 records during execution to maintain accuracy by allowing the correct blocking of records that share a common rare suffix.

For standard suffix array we compared records only located in the same exact suffix block, but for the improved suffixes we used two similarity functions the edit distance and Q-gram for construction of suffix blocks, using a threshold of 0.9 for similarity between each suffix codes this allow us to increase the size of blocks and allowing more records to be compared. The threshold value selected after several runs to ensure we have a higher number of correct duplicates.

The second experiment have been executed twice for each method, the first one we have varied the minimum suffix length parameter from 2 up to 15 while keeping the maximum block size at 50. The second one we have varied the maximum block size from 2 to 50 while keeping the minimum suffix length at 2.

For comparison of records to detect duplication we have used the same four rules used earlier in the first experiment which mainly based on similarity of surnames , given names and social security numbers .

4.4.2.1 Results Analysis

In our first run we have evaluated the effect of varying the maximum block size from 2 up to 50 while keeping the minimum suffix length of no more than 2. For standard blocking we saw that there is stability in detecting duplicates when the maximum block size is over than 5, while the standard suffix array is able to get more duplicates at this point. As shown in Table 4.4 for a block size of more than 30 we saw that both improved suffix array can enhance the detection process of at least 7% comparing to standard suffix array and this is due to the using of similarity functions to include more records in each block of suffixes. The typographical problems such as swapping or replacing of characters in records fields can be solved by using these similarity functions.

For an overall detection performance the suffix array with all of its variations have improved the detection process up to 45% with comparison to standard blocking. This can be reasoned to the nature of comparison more records in the suffixes array. The use of suffix array as a basis for blocks has increased the chance of including more records in each block. But an interesting notice that using similarity for suffix blocks creations where the

maximum block size is smaller than 30 have a side effect in decreasing the chance of detecting duplicate since it tends to expand the block sizes and causes the block to be excluded entirely from the comparisons process .

Maximum block size	Standard Blocking	Standard Suffix Array	Improved Suffix Array (Edit Distance)	Improved Suffix Array (Q-gram)
2	304	278	20	48
8	1673	2030	930	1241
13	1717	2329	1582	1831
22	1753	2587	2334	2467
30	1753	2738	2695	2735
38	1753	2786	2885	2849
44	1753	2813	2994	2929
50	1753	2866	3082	3012

Table 4.4: Total number of identified duplicates Vs the Maximum block size

Using the standard measurements of PC, PQ and RR, we can measure accuracy and scalability of the compared methods. For PQ or precision we can see that a block size of less than 10 is almost the same in all methods. But after that point the quality of correctly identifying duplicates is decreases with comparison to standard blocking and this can be reasoned to the fact that more diverse and irrelevant records are included in each block and the total number of blocks is increases and each block has records from a wide range of sources.

For PC we can see stability in detecting records for standard blocking when the maximum block size of more than 14 and due to the fact that most of the blocks size in the data set don't exceed that number. But for the suffix array methods we can see an improvement in detecting more true matches since we have more blocks and same record is now located in more than one block and thus increasing the possibility of being identified as a duplicate by the detection process.

For RR all methods give almost similar results between 100% up to 99.990% where this ratio in the fever of the suffix array since it has detected more possible duplicate with comparison to the standard blocking.

The F-Measure which calculate the mean of RR and PC and measure the accuracy of our tests is showing the F-Measure increasing whenever moving from standard blocking through to the improved suffix method and almost similar results for all suffix methods.

In our second experiment we have evaluated the effect of varying the minimum suffix length from 2 up to 15 while keeping the maximum block size of no more than 50. We can see that always under these circumstances that improved suffix array will result in high number of duplication identification whenever suffix lengths is low and this will result in an increasing number of blocks and therefore the number of comparison. Increasing the suffix length value will cause more records not to be compared since its BKV length is less than the minimum suffix length. For standard blocking decreasing the minimum suffix length (BKV length) will decrease the number of identified duplicates especially whenever the minimum suffix length is greater than 7 and due to the fact that most of the blocking key values are usually less than this number and will drop them from being taken in consideration during comparison. The processing time calculated for this experiment is always showing a decrease in time needed whenever we increase the minimum suffix length since more and more records is less compared.

The PQ for the second experiment shows a stability in value for standard blocking for all of the blocking key value lengths since every record will be compared only in one block and this will decrease the chance of being wrongly compared. For the suffix array method PQ

increases to some extent whenever we increase the minimum suffix length and that can be reasoned for a simple fact that records of larger suffix lengths will only located in a less number of blocks and this will decrease the chance of being wrongly compared with irrelevant records.

For PC the same reasons of increasing the suffix length value will cause more records not to be compared since its BKV length is less than the minimum suffix length and this will reduce the chance of identifying true possible duplicates and this is also applicable to the standard blocking method.

For RR all methods have almost similar results between 100% up to 99.990% where this ratio in the fever of the suffix array since it has detected more possible duplicates with comparison to the standard blocking. But all methods have almost the same result whenever suffix lengths value greater than 5-7 characters.

Measuring the time efficiency for these methods, the processing time collected during the tests shows clearly that the improved suffix array method has the highest processing time among all other methods, this can be reasoned by the high number of comparisons performed in each block and the overhead time needed when using of similarity functions for comparing each suffix with each neighbor suffix. The standard blocking always has the lowest processing time since we have only small number of comparisons done in each block and the elimination of suffix formation for creation of blocks.

Experiment 2 results graphs are shown in appendix 3.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

This chapter summarizes the conclusions we have achieved in this thesis and discusses possible future works that can further extend or improve our work.

5.1 Conclusions

Duplicate detection which aims to identify different or multiple records that refer to one unique real world entity or object is a crucial step for preparing data of high quality and can increase the information availability in many application areas.

In this thesis we have proposed a generic framework for duplicate detection process, provided four simple brief algorithms for blocking methods and their variations, and we have experimentally compare their performance and efficiency.

Using a synthetic same data set, we have implemented the standard blocking, sorted neighborhood method and the sorted blocks methods using an equal block size. The experiments also confirmed that using the sorted neighborhood method can make the duplicate detection process more efficient by detecting more duplicates without the need to compare all records which is very crucial for many systems performance.

Result from the experiments confirmed that standard blocking can be enhanced by using overlapping partitions where we can see that by using them, we were able to detect more duplicates records.

Our experiments also confirmed that the generalized algorithm of sorted blocks where overlapping block size is equal to block size -1 can slightly outperform the windowing

method especially in term of the percentage of true duplicate record pairs detected by the sorted blocks to the total number of correctly matched pairs exist in the data set (pair completeness) but in the expense of processing time needed.

The experiment also shows the effect of varying the value for threshold and its effect in the measurements percentages. Choosing the proper threshold depends in our interest, where we would like to detect a large number of true matches without compromising the quality.

Also using the same data set and decision model we have implemented the standard blocking method and the standard suffix array method and also proposed two variations to it, the improved suffix array using edit distance and the improved suffix array using Q-gram. The experiments confirmed previous results that the accuracy of standard blocking can be dramatically improved using the suffix array and its variations, deferent parameters such as the maximum block size and minimum suffix length have important effects regarding the performance, accuracy and quality of the results.

5.2 Future Work

Our work on duplicate record detection and the various methods for enhancing the efficiency of the blocking methods have confirmed previous results, they also lay basis for future work. There are a number of key challenges that are significant to address in the near future.

First, the experiment was done using synthetic data set and needed to be confirmed with different data sets, with real data sets and with large-scale data sets in order to verify the effectiveness and efficiency of the proposed blocking methods.

Second, many parameters have been varied manually, and this is usually done with the help of an expert from the problem domain which open the door for the need to be adjusted automatically by algorithms during the processing time in an adaptive way to achieve better results. Parameters such as threshold values are one of the major challenging values that show a promising possible future work.

Third, in this thesis we have implemented the blocking key value that have been used for sorting the data set in different ways using phonetic coding or by concatenating different record attribute and there is more work needed to address their effect in obtaining optimized result for blocking.

Fourth, in this thesis we have implemented the decision model using a Rule-Based approach where in fact there are different approaches used for decisions that rely on training data to "learn" how to match the records. This category includes probabilistic approaches and supervised machine learning approaches that can be used to enhance the accuracy and quality of the results.

Fifth, our work in this thesis in term of performance measurements have focused on processing time as a major measurement where in the future we can address other needed criteria such as the effect on systems resources such as memory and disk usage .

Finally, we need to have a comprehensive framework that can determine which blocking method is better with specific data set, and that requires evaluating and comparing other blocking methods that we didn't implement in this thesis such as the Q-gram based blocking and Canopy clustering.

REFERENCES

Aizawa, A. & Oyama, K. (2005): A Fast Linkage Detection Scheme for Multi-Source Information Integration, Proceedings of the International Workshop on Challenges in Web Information Retrieval and Integration, p.30-39.

Baxter, R. , Christen, P. & Churches, T. (2003): A Comparison of Fast Blocking Methods for Record Linkage, *ACM* SIGKDD Workshop on Data Cleaning, Record Linkage and Object Consolidation, Washington DC, pp. 25--27.

Bertolazzi, P., DeSantis, L. , & Scannapieco, M. (2003) , Automatic Record Matching in Cooperative Information Systems, Proceedings of the Workshop on Data Quality in Cooperative Information Systems (ICDT'03).

Bilenko, M. & Mooney, R. (2003) ,Adaptive Duplicate Detection Using Learnable String Similarity Measures. Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2003), Washington DC, pp.39-48.

Black, P. (2008). Levenshtein distance, available http://www.itl.nist.gov/div897/sqg/dads/HTML/Levenshtein.html, Accessed April 2010.

Bleiholder, J. & Naumann, F. (2008) : Data Fusion , Data Fusion Journal : ACM Computing Surveys, Vol. 41, No. 1, Article 1.

Chaudhuri, S. , Ganjam, K. ,Ganti, V. & Motwani, R. (2003) Robust and Efficient Fuzzy Match for Online Data Cleaning. ACM SIGMOD International Conference on Management.

Christen, P. & Churches, T. (2005). A Probabilistic Deduplication, Record Linkage and Geocoding System. Advances in Data Mining: Theory, Methodology, Techniques, and Applications. State-of-the-Art Lecture Notes in Artificial Intelligence, Volume 3755, Springer-Verlag.

Christen, P. (2006) , A Comparison of Personal Name Matching: Techniques and Practical Issues. Joint Computer Science Technical Report Series, September. Tr-Cs-06-02, Department Of Computer Science, Computer Sciences Laboratory Research School Of Information Sciences And Engineering, The Australian National University .

Christen, P. (2007) Improving data linkage and deduplication quality through nearest-neighbour based blocking. Proceeding of thirteenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'07) .

Christen, P. & Goiser, K. (2007) Quality and Complexity Measures for Data Linkage and Deduplication. Quality Measures in Data Mining ,Studies in Computational Intelligence , Volume 43/2007 .

Cibella, N. , Fortini, M. , Scannapieco, M. , Tosco, L. & Tuoto, T. (2008) Theory and practice of developing a record linkage software. Proceeding of the Workshop "Combination of surveys and administrative data" of the CENEX Statistical Methodology Project Area "Integration of survey and administrative data". Vienna .

Cohen, W. (1998) Integration of Heterogeneous Databases Without Common Domains Using Queries Based on Textual Similarity. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD'98).

Cohen, W. , Ravikumar, P. & Fienberg, S. (2003) A comparison of string distance metrics for name-matching tasks, Proceedings of the IJCAI -2003 Workshop on Information .

De vries, T. , Ke , H. & Chawla, S. (2009), Record Linkage in the Industry: Applications of an Improved Suffix Array Blocking Method.Techincal Report, School Of Information Technologies,The University Of Sydney .

Draisbach, U. & Naumann, F. (2009): Comparison and generalization of blocking and windowing algorithms for duplicate detection, Proceedings of QDB 2009 Workshop at VLDB.

Elfekey, M. , Vassilios, V. , & Elmagarmid, A. (2002) , “TAILOR: A Record Linkage Toolbox,” IEEE International Conference on Data Engineering 2002, San Jose, USA,

Elmagarmid, A. , Ipeirotis, P. & Verykios, V. (2007) Duplicate Record Detection: A Survey, IEEE Transactions On Knowledge And Data Engineering, VOL. 19, NO. 1.

Gu, L. , Baxter, R. , Vickers, D. & Rainsford, C. (2003) Record Linkage: Current Practice and Future Directions : Technical Report 03/83, CSIRO Mathematical and Information Sciences.

Gu, L. & Baxter, R. (2004) Adaptive Filtering for Efficient Record Linkage , 2004 SIAM Int. Conf. on Data Mining , Orlando, Florida .

Hernández, M. & Stolfo, S. (1998), Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem, Data Mining and Knowledge Discovery ,Volume 2 , Issue 1 , Pages: 9 – 37 .

Idmatchsystems.com (2010) Soundex Problems & Issues, Available http://www.idmatchsystems.com/products/idmatch/soundex-problems. Accessed April , 2010.

Innerhofer-Oberperfler , R. (2004) Using Approximate String Matching, Techniques to Join Street Names of Residential Addresses, Bachelor Thesis ,Academic Year 2003/2004 1st Graduate Session .

Jurczyk, P. , Lu , J. , Xiong, L. , Cragan, J. & Correa, A. (2008) FRIL: A Tool for Comparative Record Linkage. Available http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2656092/?log%24=activity , AMIA Annu Symp Proc. 2008; 440–444. Published online 2008. , Accessed June 2010 .

Lait, A. & Randell, B. (1998) An Assessment of Name Matching Algorithms , Society of Indexers Genealogical Group, Newsletter Contents, SIGGNL issues 17.

Müller, H. & Freytag, J. (2003) Problems, Methods, and Challenges in Comprehensive Data Cleansing Technical Report. Humboldt University Berlin , Available http://www.dbis.informatik.hu-berlin.de/fileadmin/research/papers/techreports/2003-hub_ib_164-mueller.pdf . Accessed May 2010.

Scannapieco, M. , Missier, P. & Batini, C. (2005) :Data Quality at a Glance : Databank Spektrum Journal issue 14 .

Shahri, H. & Shahri, S. (2006) Eliminating Duplicates in Information Integration: An Adaptive, Extensible Framework, IEEE Intelligent Systems, Volume 21, and Issue 5, Pages: 63 – 71.

Tamilselvi, J. & Saravanan, V. (2009) Detection and Elimination of Duplicate Data Using Token-Based Method for a Data Warehouse: A Clustering Based Approach, International Journal of Dynamics of Fluids , ISSN 0973-1784 Volume 5, Number 2, pp. 145–164

Tang, Y. , Zhang, J. , Tan, C. & Wong, M. (2003) A Five-step Approach to Multi-source Information Infusion with Guaranteed Data Integrity. Singapore Institute of Manufacturing Technolog,Available http://www.simtech.a-star.edu.sg/research/technicalreports/tr0302.doc . Accessed May ,2010.

Ukkonen, E. (1992) ,Approximate string-matching with Q-grams and maximal matches ,Theoretical Computer Science , Volume 92 , Issue 1 ,Selected papers of the Combinatorial Pattern Matching School ,Pages: 191 - 211 .

Wang, X. (2008) Matching records in multiple databases using a hybridization of several technologies, Dissertation. Department of Industrial Engineering. University of Louisville, KY,USA .

Yan, S. , Lee ,D. , Kan ,M. & Giles, C. (2007) Adaptive Sorted Neighborhood Methods for Efficient. Record Linkage. International conference on digital libraries JCDL'07, Vancouver, British Columbia, Canada.

Yancey, W. (2007) BigMatch:A Program for Extracting Probable Matches from a Large File , Research Report Series , (Computing #2007-1) , Statistical Research Division ,U.S. Census Bureau.

Ziegler, P. & Dittrich, K. (2004) Three Decades of Data Integration - All Problems Solved, In 18th IFIP World Computer Congress, Building the Information Society.

APPENDICES

Appendix 1 Blocking Algorithms Codes

1.A: Standard blocking algorithm using equal block size

```
Input: Data set (table), block size
Output: LOG table
Var: n ← no. of records, currentRow, lastrecord, innerRow, outerrow
Begin
Sort table using key
currentRow=1
while( currentRow<= n)
        lastrecord =minimum (currentRow+ blockSize-1,n)
        while (outerRow< lastrecord )
              Get record (outerRow)
              innerRow=outerRow+1
              while (innerRow<=  lastrecord )
                 Get record (innerRow)
                 Compare innerrow record with outerrow record using decision rules
                 If matched
                          Log duplicate
                 End if
                 innerRow = innerRow +1
             endwhile
            outerRow= outerRow +  1
         endwhile
       currentRow=currentRow+ blockSize
endwhile
End
```

1.B : Sorted Neighborhood Algorithm using equal block size

```
Input: Data set (table), windowSize
Output: LOG table
Var: n ← no. of records, currentRow, lastRowInWindow
Begin
Sort table using key
 lastRowInWindow= minimum (windowSize,noOfRecords)
 currentRow=1
   while ( currentRow<= n )
          Get record (currentRow)
          windowlistrow=currentRow+1
          while (windowlistrow<=  lastRowInWindow)
             Get record (windowlistrow)
             Compare currentRow record with windowlistrow record using decision rules
              If matched
                     Log duplicate
              End if
             windowlistrow = windowlistrow+1
           endwhile
           lastRowInWindow= minimum (lastRowInWindow+1,noOfRecords) ;
           currentRow=currentRow+1
    endwhile
 End
```

1.C Sorted Blocks Algorithm using equal block size

```
Input: Data set (table), block size
Output: LOG table
Var: n ← no. of records, currentRow, lastrecord, innerRow, outerrow
 Begin
 Sort table using key
Overlap = blockSize /2
 currentRow=1
  while( currentRow<= n )
  lastrecord =minimum (currentRow+ blockSize-1,n)
  while (outerRow< lastrecord )
                Get record (outerRow)
                innerRow=outerRow+1
                while (innerRow<=  lastrecord )
                     Get record (innerRow)
                     Compare innerrow record with outerrow record using decision rules
                     If matched
                      Log duplicate
                     end if
                    innerRow = innerRow +1
                 endwhile
                outerRow= outerRow +  1
   endwhile
  outeroverlap = (currentRow+blockSize- overlap)
  while (outeroverlap  <=lastrecord)
         Get record (outeroverlap  )
          inneroverlap = currentRow+blockSize
          lastrecordoverlap= minimum  (noOfRecords, outeroverlap+ overlap)
          while (inneroverlap<=lastrecordoverlap )
               Get record(inneroverlap)
                Compare inneroverlap record with outeroverlap record using decision rules
                If matched
                   Log duplicate
                End if
               Inneroverlap= inneroverlap+1
           endwhile
           outeroverlap  = outeroverlap  +1
    endwhile
    currentRow=currentRow+ blockSize
endwhile
End
```

1.D: Suffix Array Algorithm

```
Input: Data set (table), min_suff_length , Max_block_size, similiarty_Threshold
Output: suffix_array , LOG table
Var: first_row_in_block,last_row_in_block,last_row_in_block, first_suffix , next_suffix
Var: no_of_records_in_block, outerRow, innerRow
Begin
 Create the suffix array inverted index
 Sort the suffix array alphabetically
 get (first suffix code)
  first_row_in_block= 1
 last_row_in_block=1
 first_suffix = get (suffix_code)
no_of_records_in_block=1
 get ( next suffix code )
while (not end of suffix array)
       next_suffix =  get(suffix_code)
       result =(calculate similarity between first_suffix and next_suffix codes)
        /* the next if statement only for improved suffix array otherwise should  equal to 1 */
       if (result >= similiarty_Threshold )
            /* only for improved suffix array otherwise does not affect */
           first_suffix=Next_suffix ;
           last_row_in_block=current row number
           no_of_records_in_block=no_of_records_in_block+1 ;
      else
          if (no_of_records_in_block>1  and  no_of_records_in_block <=Max_block_size )
             outerRow=first_row_in_block
             while (outerRow< last_row_in_block)
                  Get record (outerRow)
                   innerRow=outerRow+1
                   while (innerRow<=  last_row_in_block))
                       Get record (innerRow)
                       Compare innerrow record with outerrow record using decision rules
                        If matched
                             Log duplicate
                         End if
                         innerRow = innerRow +1
                     endwhile
                    outerRow= outerRow +  1
                 endwhile
            endi if
            first_row_in_block= current row number
            first_suffix =  get(suffix_code)
            last_row_in_block== current row number
            no_of_records_in_block=1
```

```
    endif
    get( next suffix code  )
 endwhile
End
```

Appendix 2: Experiment 1 Results Graphs

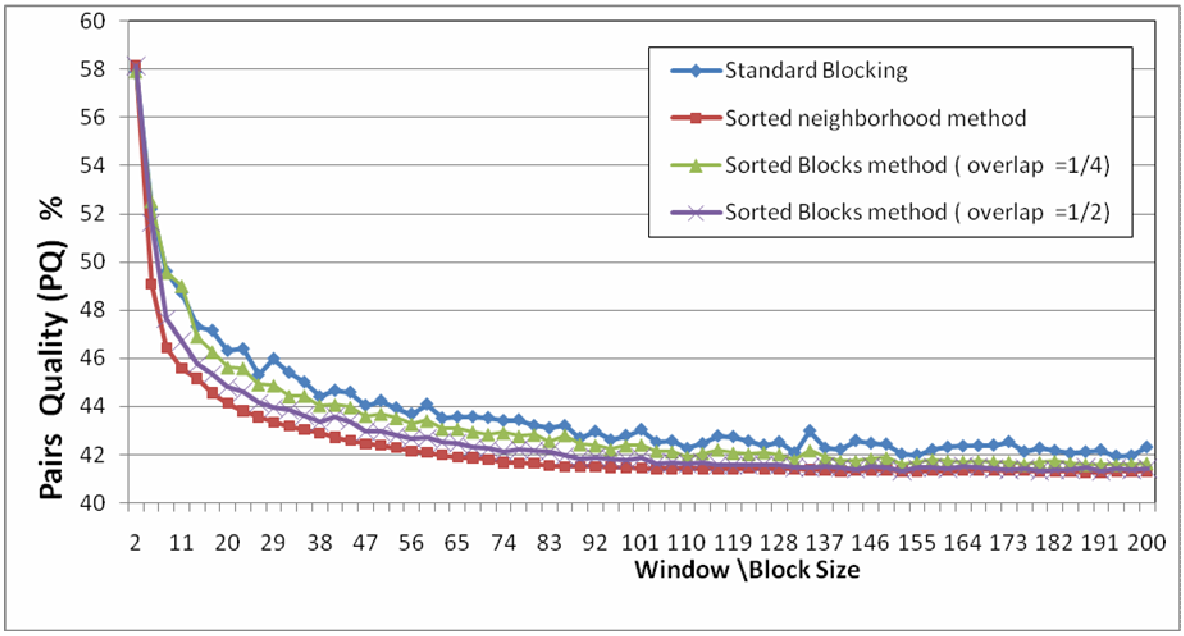

Figure A.2.1: PQ Vs window/ block size

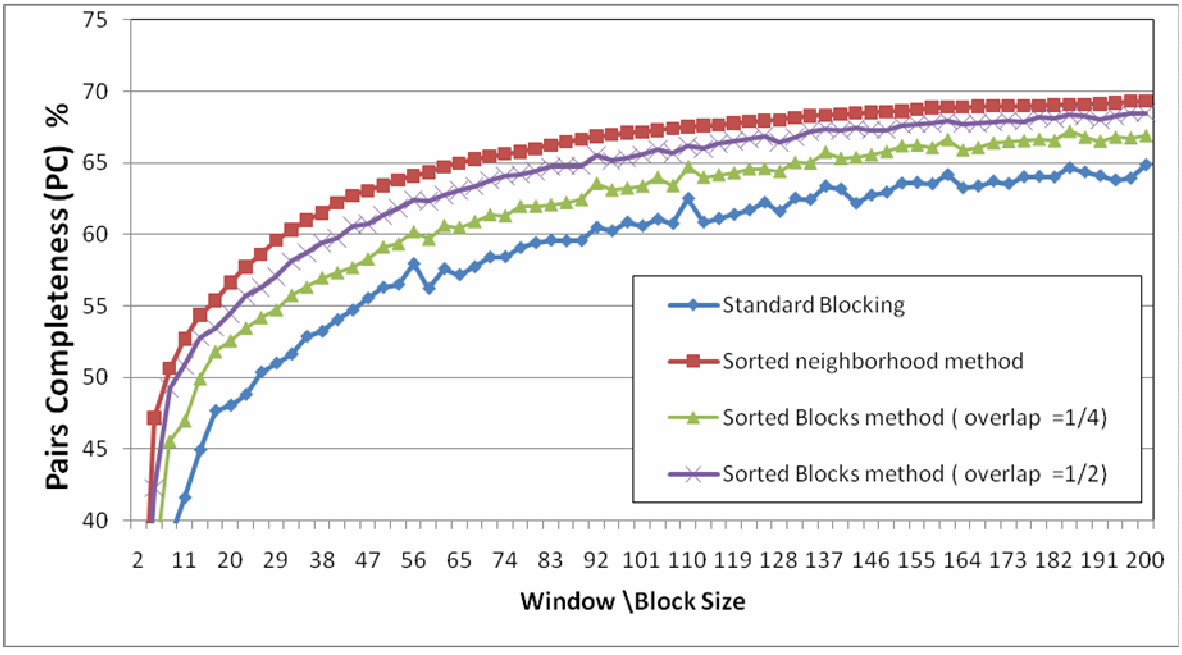

Figure A.2.2: PC Vs window/ block size

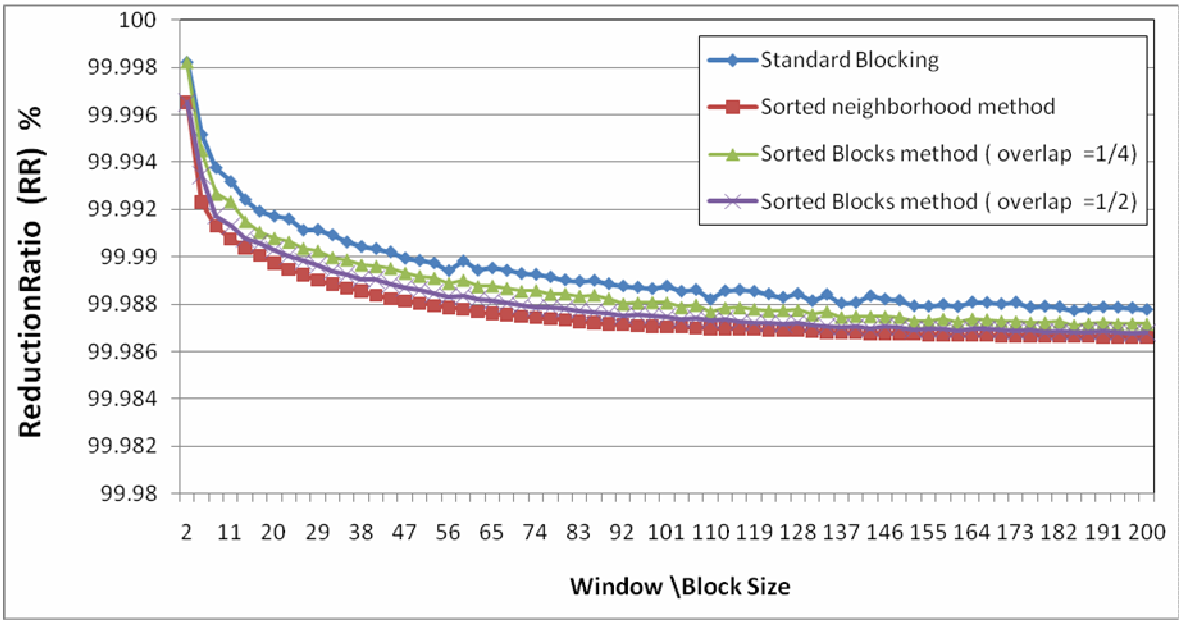

Figure A.2.3: RR Vs window/ block size

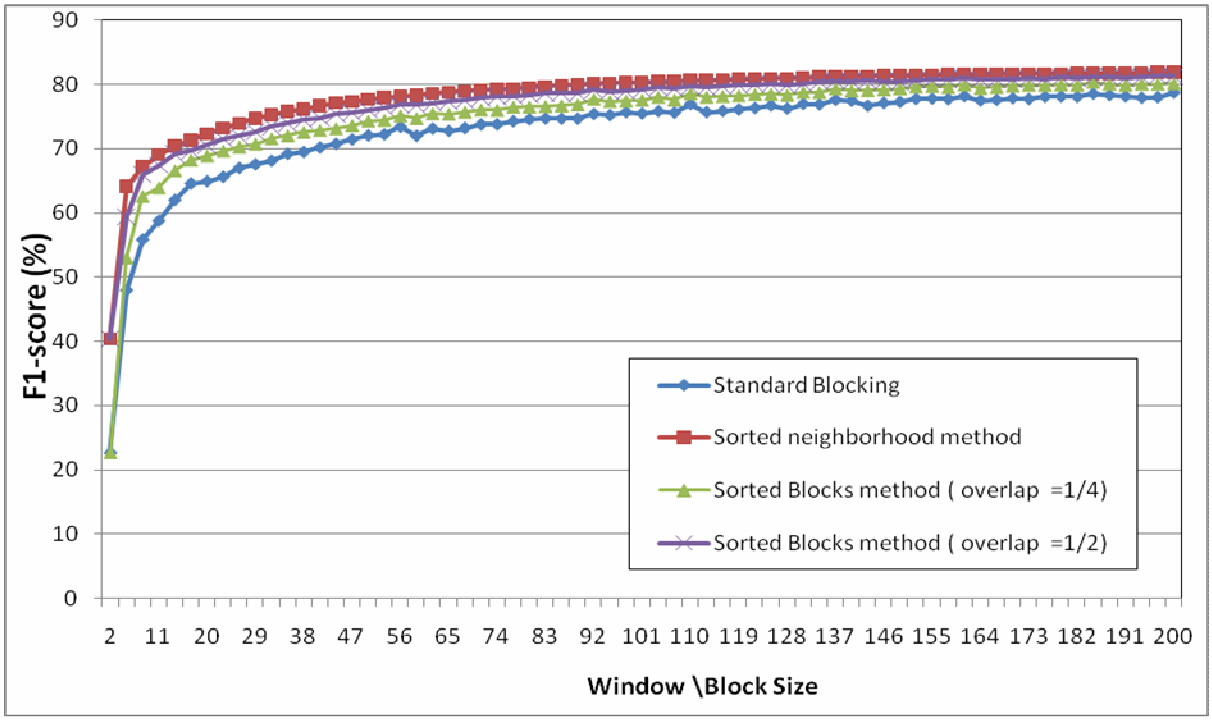

Figure A.2.4: F1-score Vs windows/ block size

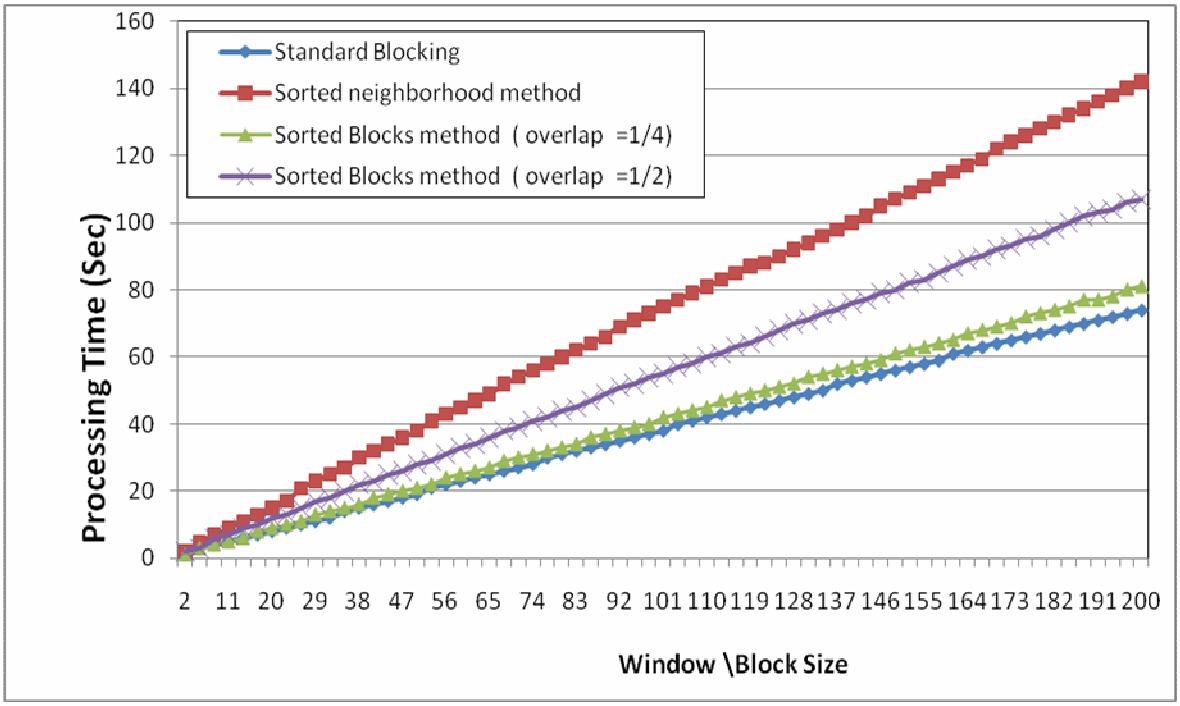

Figure A.2.5: Scalability of the blocking methods in respect to window/ block size

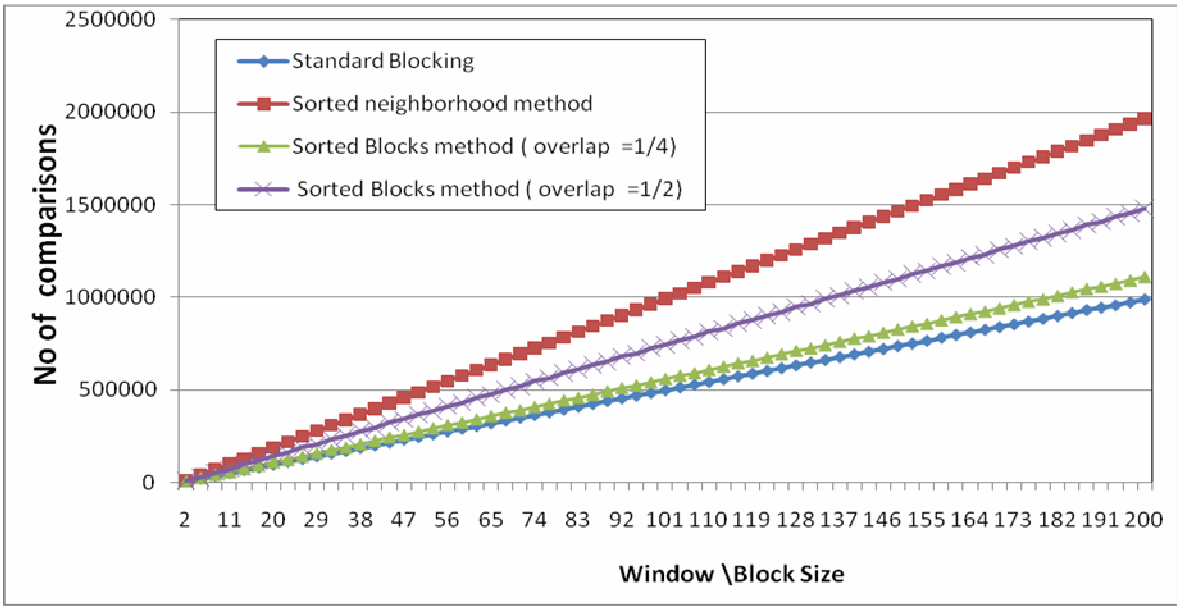

Figure A.2.6: Complexity comparison of the blocking methods in respect to window/ block size

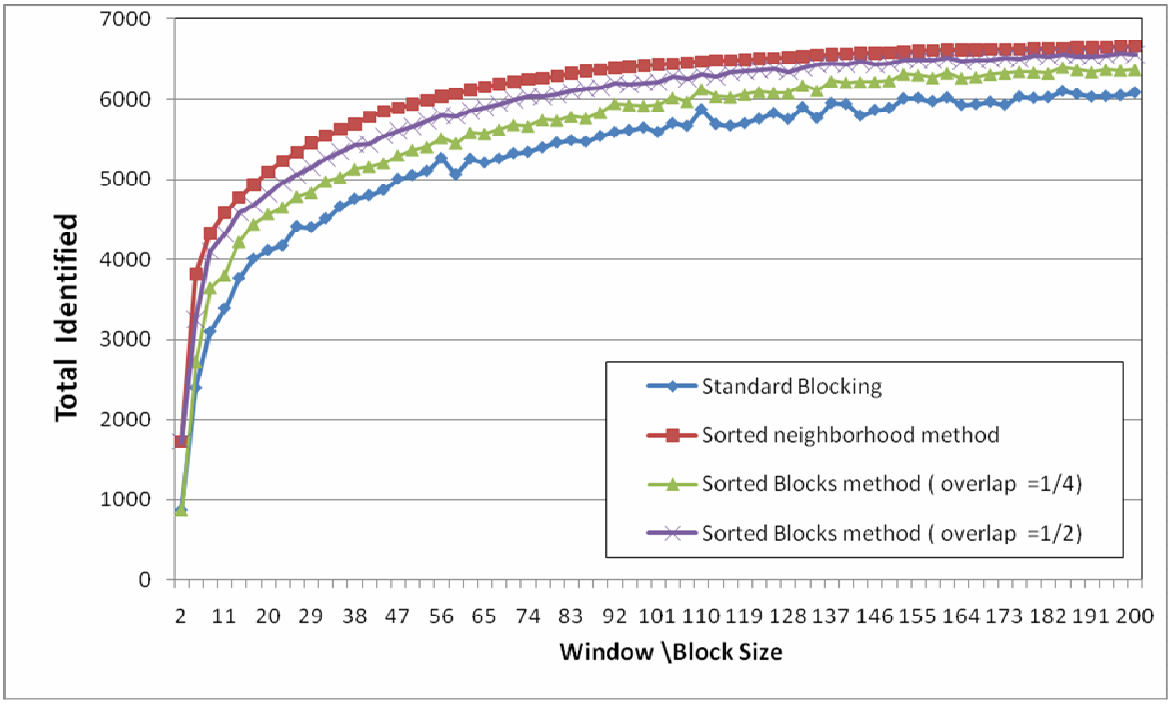

Figure A.2.7: Total number of identified duplicate Vs windows/ block size

Appendix 3: Experiment 2 Results Graphs

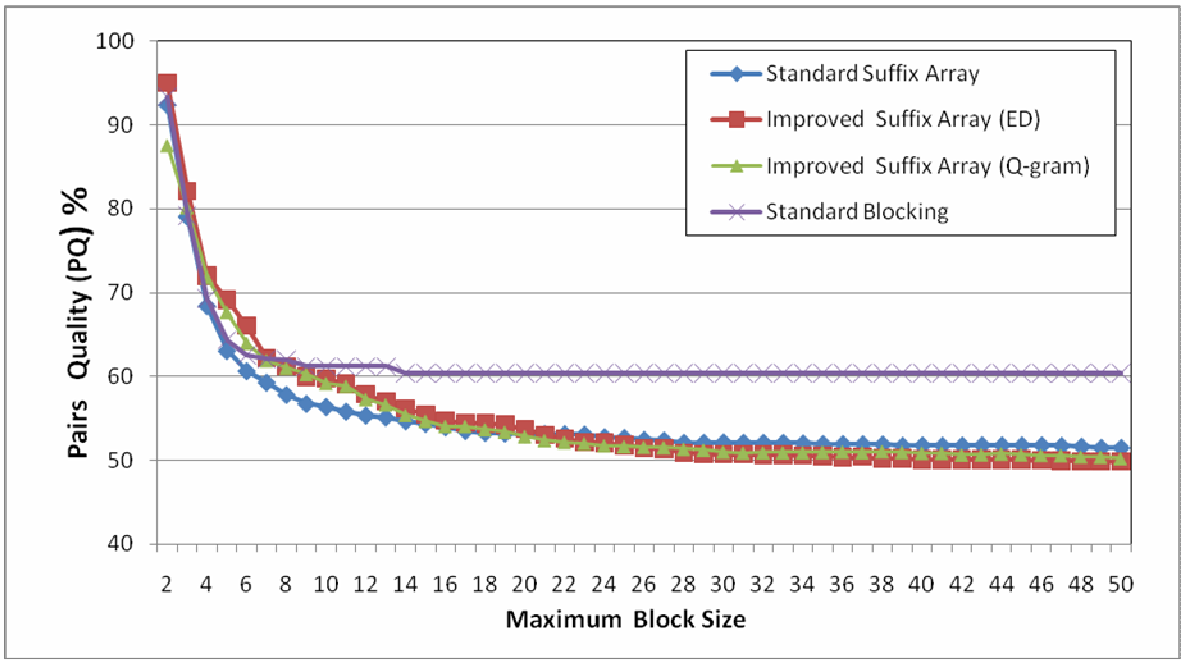

Figure A.3.1: PQ Vs the maximum block size

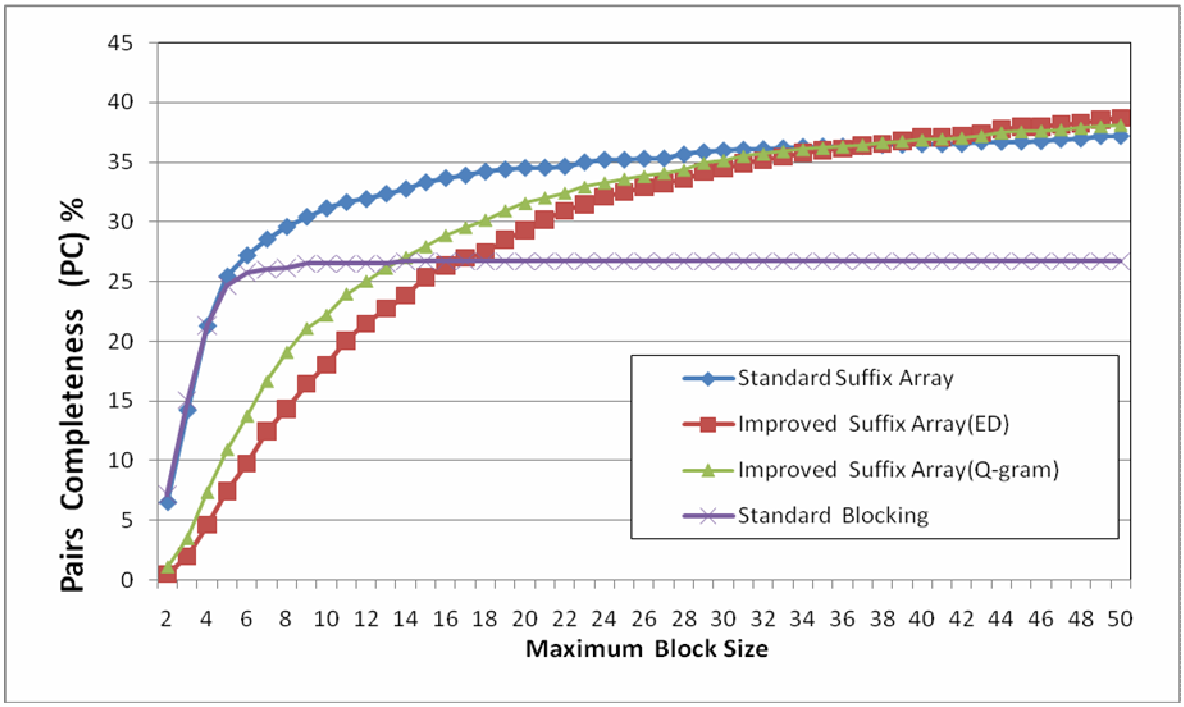

Figure A.3.2: PC Vs the maximum block size

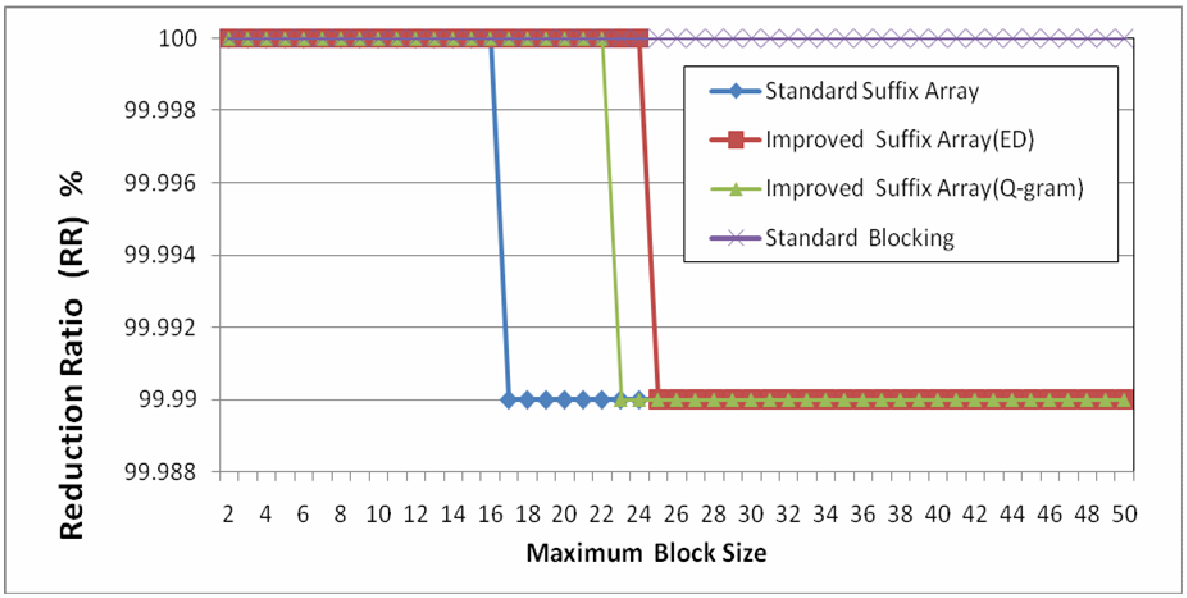

Figure A.3.3: RR Vs the maximum block size

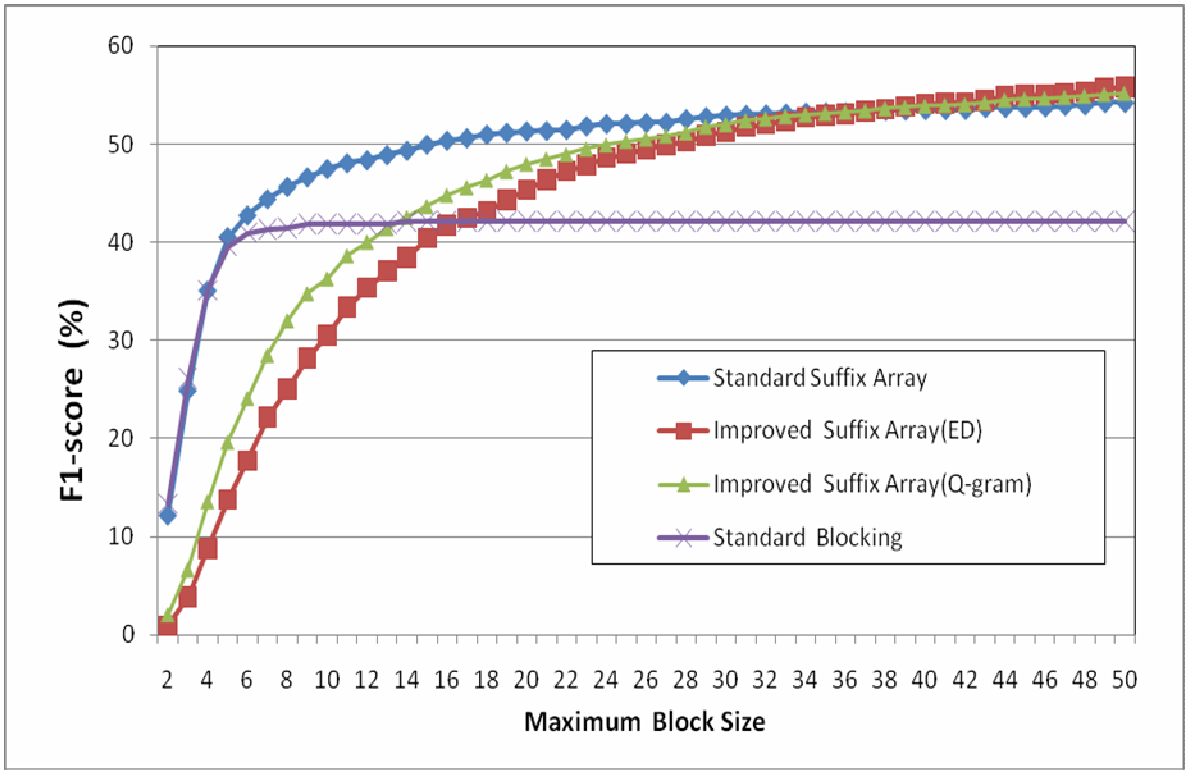

Figure A.3.4: F1-score Vs the maximum block size

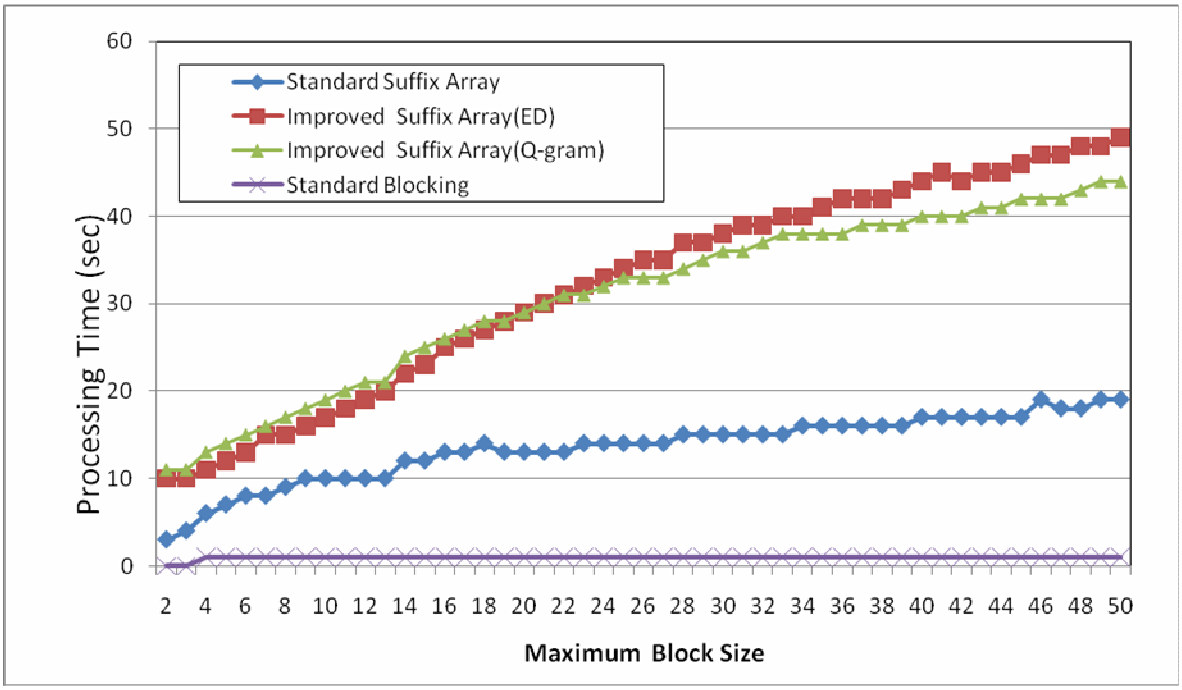

Figure A.3.5: Scalability of the blocking methods in respect to the maximum block size

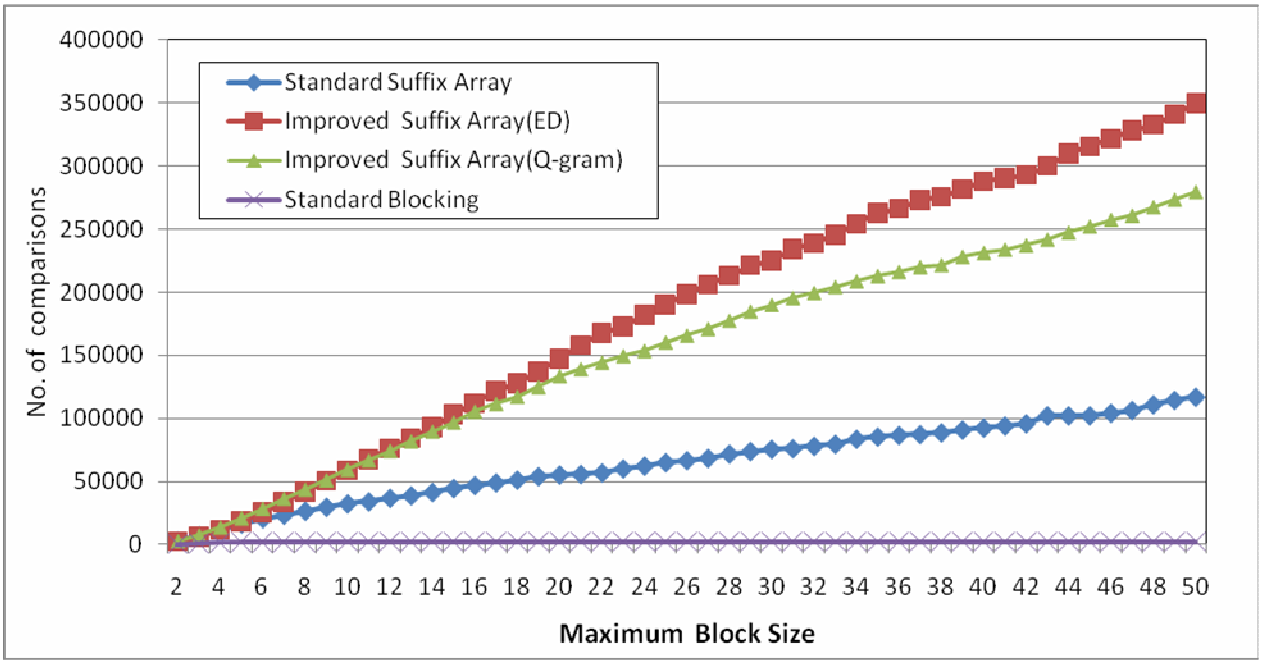

Figure A.3.6: Complexity comparison of the blocking methods in respect to the maximum block size

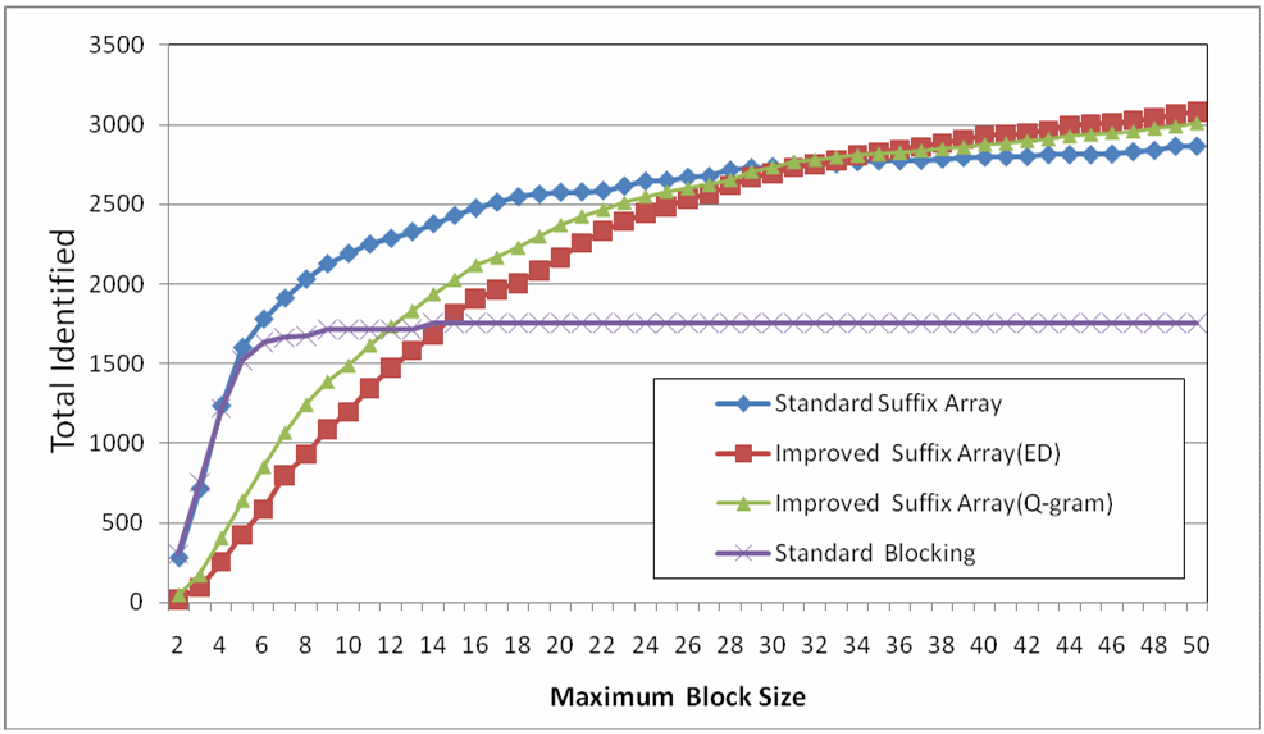

Figure A.3.7: Total number of identified duplicates Vs the maximum block size

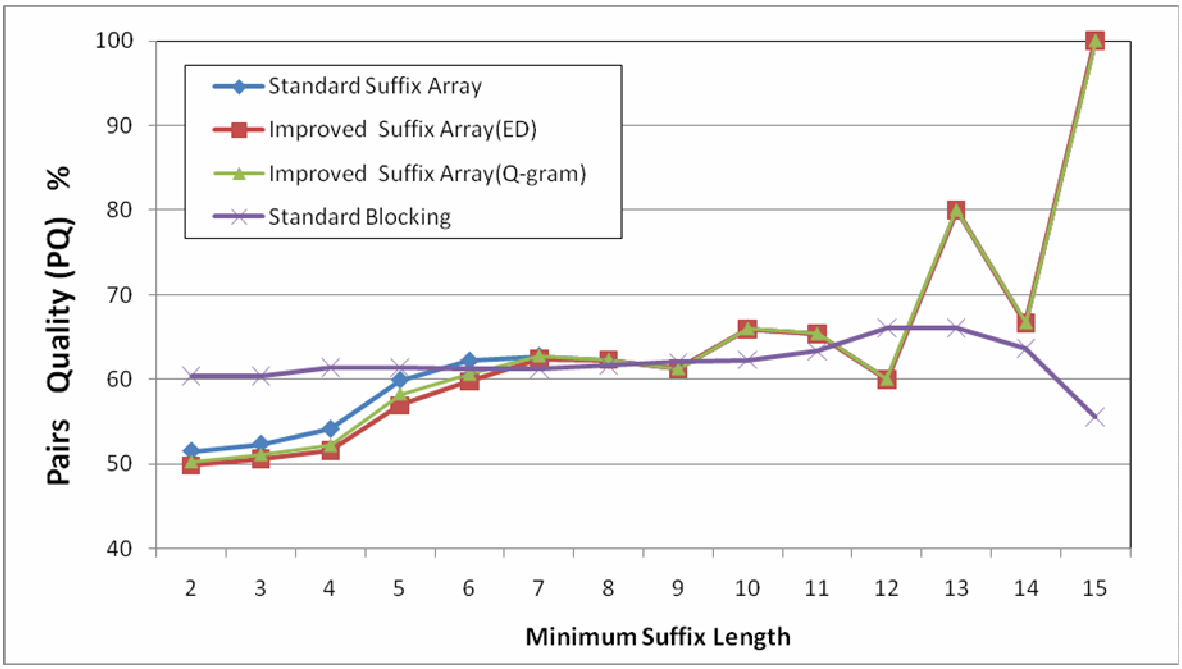

Figure A.3.8: PQ Vs the minimum suffix length

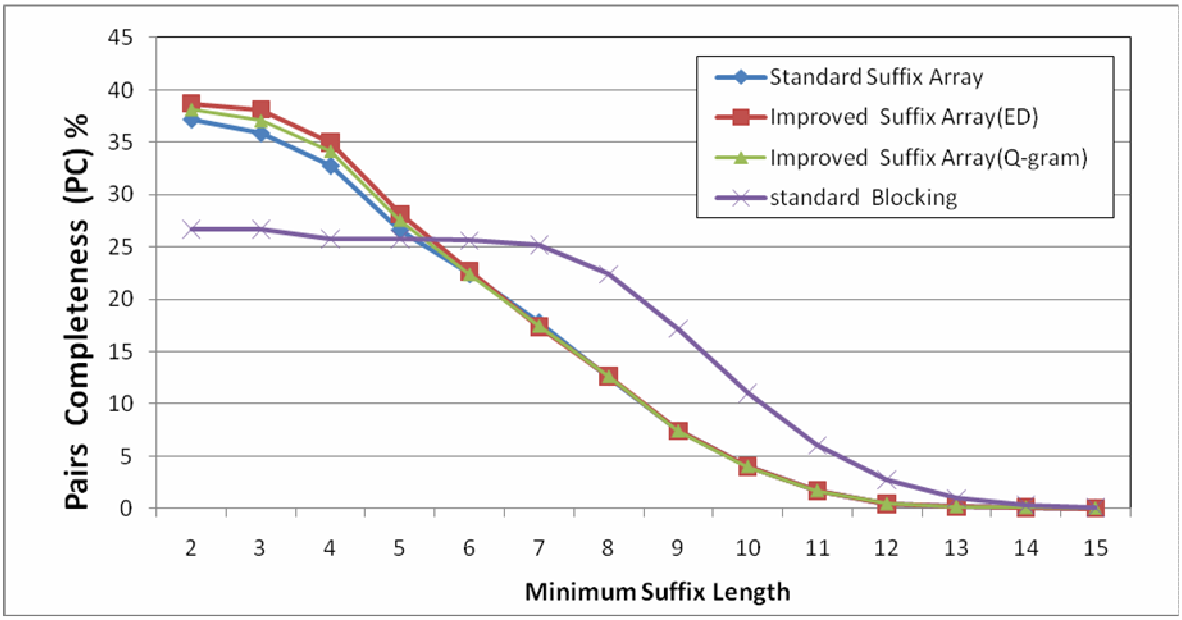

Figure A.3.9: PC Vs the minimum suffix length

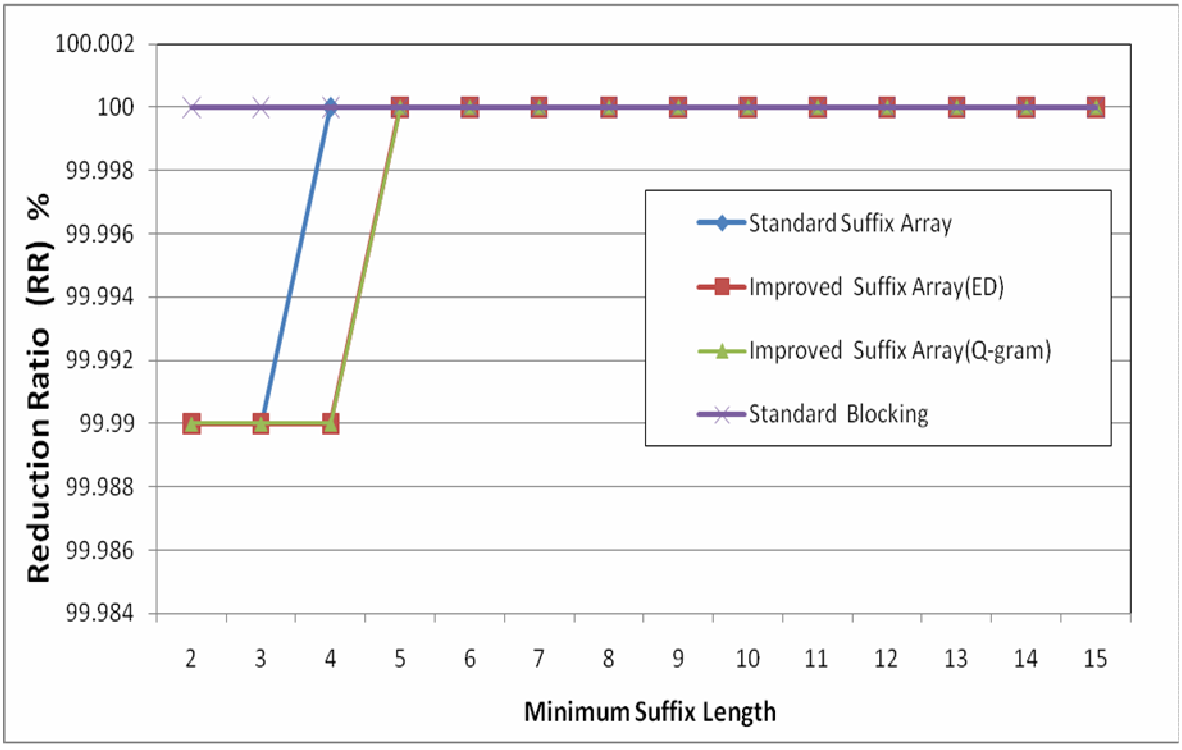

Figure A.3.10 RR Vs the minimum suffix length

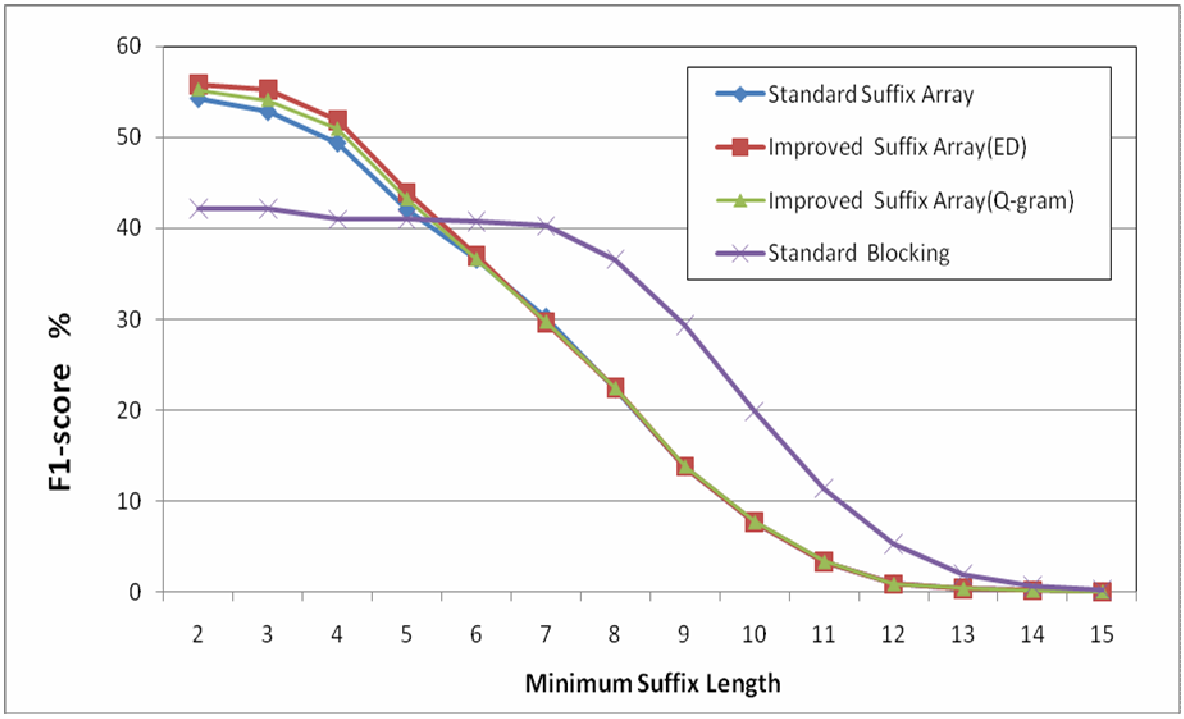

Figure A.3.11: F1-score Vs the minimum suffix length

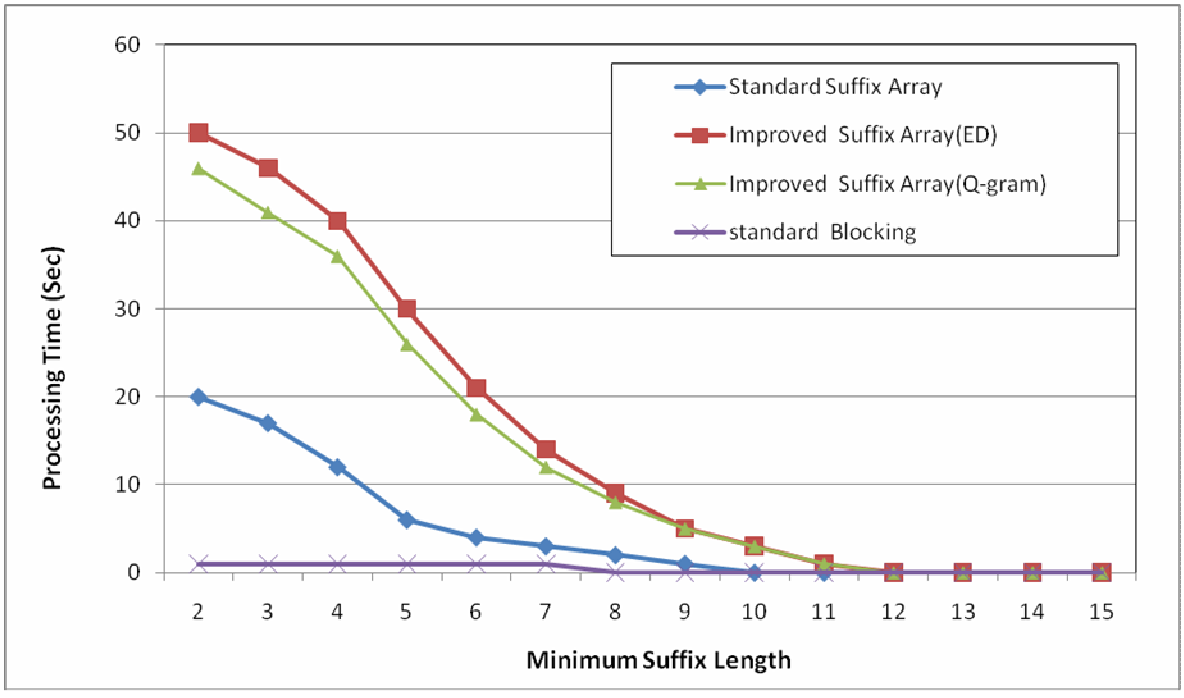

Figure A.3.12: Scalability of the blocking methods in respect to the minimum suffix length

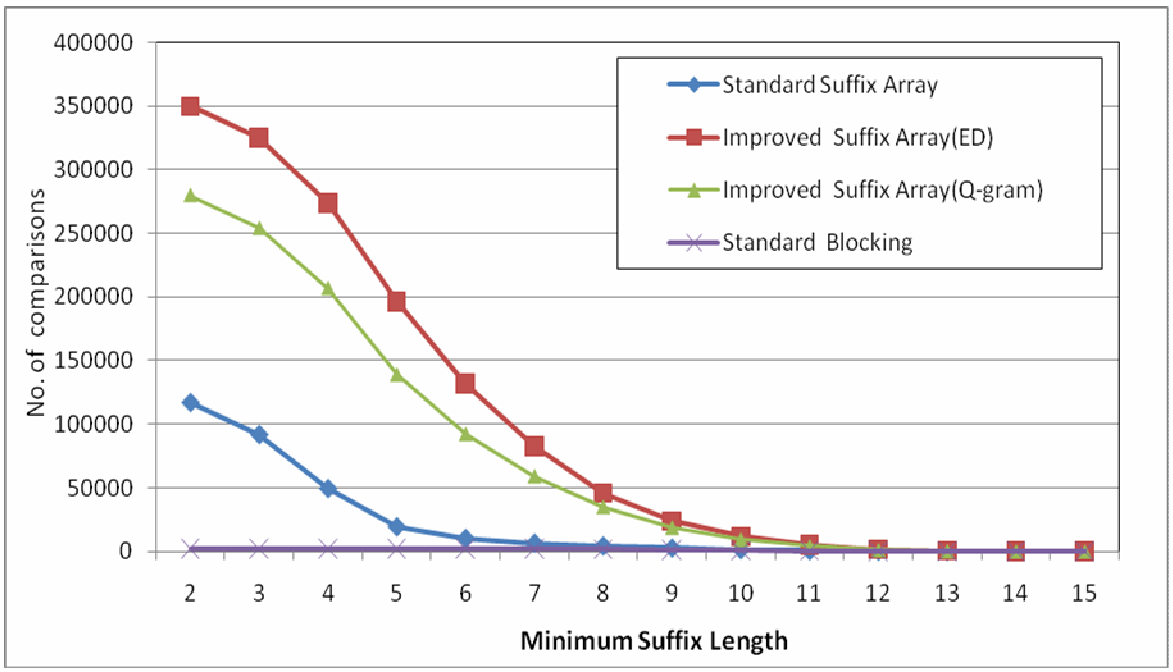

Figure A.3.13: Complexity comparison of the blocking methods in respect to the minimum suffix length

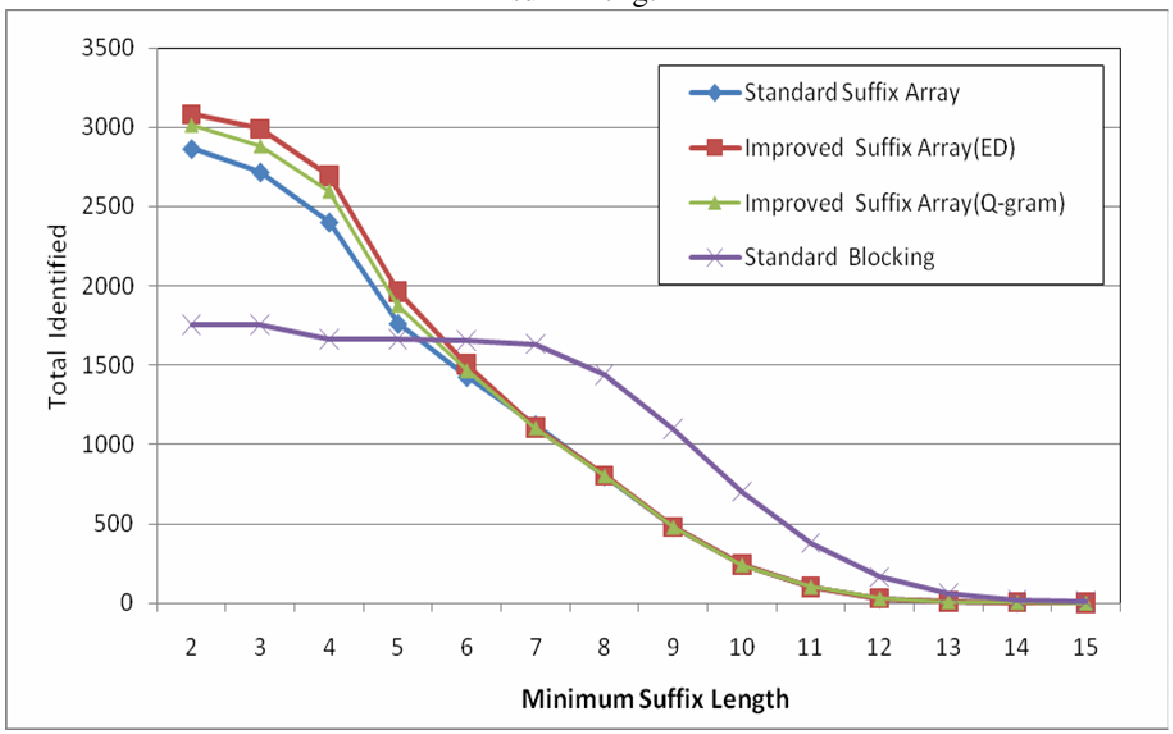

Figure A.3.14: Total number of identified duplicates Vs the minimum suffix length