A Comparative Study of Record Matching Algorithms

Shirley Ong Ai Pei (276905)

European Master in Informatics (EuMI)
Media Informatics
RWTH Aachen, Germany and University of Edinburgh, Scotland
2008

Abstract

Record matching is an important process in data integration and data cleaning. It involves identifying cases where multiple database entities correspond to the same real-world entity. Often, duplicate records do not share a common key and contain erroneous data that make record matching a difficult task. The quality of a record matching system highly depends on a good approach that is able to accurately detects duplicates in an efficient and effective way. Despite the many techniques that have been introduced over the decades, it is unclear which technique is the current state-of-the-art. Hence, the objectives of this project are:

1. Compare a few record matching techniques and evaluate their advantages and disadvantages.

2. Develop a technique that combines the best features from these techniques to produce an improved record matching technique.

Currently, there are two main approaches for duplicate record detection, categorised into approaches that rely on training data, and approaches that rely on domain knowledge or distance metrics. This project focuses on comparisons between the Probabilistic-based approach from the former category, and the Rule-based approach from the latter category. For the Probabilistic-based approach, instead of relying on training data, we employed the Expectation Maximization (EM) algorithm to find maximum likelihood estimates of parameters in the Probabilistic models. Our experimental study reveals that the Probabilistic-based approach, employing the EM algorithm, yields better results than the Rule-based approach when the requirements to generate the Probabilistics parameters are satisfied.

Another reason for the poor results from the Rule-based approach is due to the lack of domain knowledge when manually crafting the ruleset. We believe that the Rule-based approach may perform well, given substantial manual effort and time to fine-tune the rules. However, extensive manual analysis of the dataset is not desirable.

To this end, we propose an approach that leverages statistics computed from the EM algorithm to automatically derive a better ruleset for the Rule-based approach. Our experimental study shows that the proposed approach performs very well, even exceeding the performance of the Probabilistic-based approach.

Acknowledgements

I would like to take this opportunity to sincerely thank the following people. The success of this project would not have been possible without them:

- Supervisors at the University of Edinburgh: Prof. Wenfei Fan and Dr. Xibei Jia
- Supervisor at Rheinisch-Westfälische Technische Hochschule Aachen (RWTH Aachen University): Prof. Matthias Jarke

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Shirley Ong Ai Pei (276905)*)

Table of Contents

1 Introduction **6**

2 Theoretical Background **10**

2.1 Comparison Subspaces 10

2.1.1 Blocking Technique 10

2.1.2 Sorted Neighbourhood Method 11

2.2 Attribute Matching Techniques 12

2.2.1 Character-Based Similarity Metrics 13

2.2.2 Token-Based Similarity Metrics 13

2.3 Duplicate Records Detection 14

2.3.1 Probabilistic-Based Approach 14

2.3.2 Rule-Based Approach 19

2.3.3 Cocktail Approach 21

3 System Overview **23**

3.1 System Architecture 23

3.2 Technologies 26

3.2.1 Java Development Kit (JDK) 26

3.2.2 MySQL Database System 26

3.2.3 Java Database Connectivity (JDBC) 26

3.2.4 Waikato Environment for Knowledge Analysis (Weka) 28

3.2.5 SimMetrics 28

4 Implementation Details **29**

4.1 Candidate Key Creator 29

4.2 Comparison Space Creator 30

4.2.1 Blocks Creator 30
4.2.2 Sliding Window Creator 32
4.3 Records Pairing Creator 32
4.3.1 Pairing Records in Blocks 33
4.3.2 Pairing Records in Sliding Window 33
4.4 Matcher 35
4.4.1 Probabilistic-Based Matcher 35
4.4.2 Rule-Based Matcher 41
4.4.3 Cocktail Matcher 44
4.5 Classifier 47
4.6 Evaluator 47

5 Experimental Study 52
5.1 Experiments Set-Up 52
5.1.1 Thresholds Selections 53
5.2 Comparisons Between the Three Approaches 54
5.2.1 Experiments on Restaurant Dataset 55
5.2.2 Experiments on Cora Dataset 61
5.2.3 Experiments on DBGen Dataset 70
5.3 Comparisons with Other Works 79

6 Conclusion And Future Work 81

A Equational Theory Rules 84

Bibliography 90

Chapter 1

Introduction

Record matching, also known as merge-purge, data deduplication and instance identification, is the problem of identifying records in the same or different databases that refer to the same real-world entity. The problem has existed for decades and has become very crucial as more and more data are stored in the database systems, and there arise the needs to integrate data for decision support applications and to clean erroneous data due to human mistakes.

There are two types of concerns for record matching; the first involves structural heterogeneity and the second concerns lexical heterogeneity. Structural heterogeneity refers to the problem of matching two databases with different domain structures. For example, a customer address might be stored in the attribute 'address' in one database but represented in attributes 'street', 'city', and 'zip' in another database. Lexical heterogeneity refers to databases with similar structure but different representation of data, such as 'J. Smith' and 'Smith, John'. For this project, we are concerned with the second heterogeneity problem and assumed that structural heterogeneity has been resolved a priori.

Currently, the approaches to solve record matching problem can be broadly classified into two categories:

- Approaches using Probabilistic models [1], and Supervised/Semisupervised Learning techniques [2, 3, 4, 5, 6] that rely on training data to learn how to match records.
- Approaches such as Rule-based [7, 8, 9] and Distance-based techniques [10, 11, 12] that rely on domain knowledge or distance metrics to match records.

Due to the various techniques that have been proposed and developed over the years,

it is uncertain which is the current state-of-the-art, an issue raised by Elmagarmid et. al. [4]. Hence, the goal of this project is to compare these techniques on real-world and large-scale benchmarking datasets, and evaluate their advantages and disadvantages. Two techniques were selected each from the two categories to bring us to the second goal of the project, which is an attempt to develop a technique that combines the best features from the both worlds. The two techniques selected for this project are the Probabilistic-based approach and the Rule-based approach.

The Probabilistic-based approach is a popular solution that are supported in a few existing record linkage and consolidation softwares such as the Link King [1], LinkageWiz [2], Febrl [13], and TAILOR [14]. Basically, the Probabilistic-based approach relies on training data to compute a maximum likelihood estimate that determines if a record pair is matching or non-matching. Instead of using training data, which may not be available all the time, we depended on the unsupervised Expectation Maximization (EM) algorithm. Also, most datasets have reasonably clear structure. The EM algorithm makes use of this information to supply the maximum likelihood estimate. However, the EM algorithm only works well under the following conditions, as highlighted in [15]:

- More than 5% duplicates in the dataset.
- Matching record pairs are well separated from non-matching record pairs.
- The rate of typographical error is low.
- Sufficient attributes to compensate for errors in other fields of the record.
- The conditional independence assumption results in good classification performance.

Meanwhile, the Rule-based approach, which is also supported in the Link King software, has garnered much attention due to its potential to result in systems with high accuracy [16]. However, the high accuracy is obtained through substantial effort and time of an expert to meticulously devise a matching ruleset. It also requires an analysis of the dataset to give a better idea to the expert how to fine-tune the rules. Desirably, we want to design a good matching ruleset without manual human effort and analysis of the dataset due to the following reasons:

[1] http://www.the-link-king.com/
[2] http://www.linkagewiz.com/

- An expert may not be available all the time.
- The dataset may be private.
- Two datasets with similar domains may behave drastically different. Therefore, the ruleset manually devised for the first dataset may not be applicable to the second dataset.

Besides devising a good technique for the matching process, a record matching system should also include a technique to increase the efficiency of the system, which is necessary particularly when dealing with large datasets. Subspaces creation is a method to reduce the number of candidate record pairs to a feasible number whilst maintaining the accuracy of the system. Most of these methods such as the blocking method [17] and the Sorted Neighbourhood (SN) method [7], use candidate keys to sort the dataset before applying a block or window to limit the number of candidate record pairs. These candidate keys are often created from a combination of the dataset attributes. Consequently, it is imperative to choose the appropriate attributes in order not to miss detecting a duplicate record.

Contributions. In response to the weaknesses of the Rule-based approach, and the challenge of devising appropriate candidate keys to create subspaces, we propose an approach, which we referred to as the Cocktail approach in this project. This approach utilises information from the EM algorithm to:

- Determine which attributes of the dataset are significant and insignificant. The insignificant attributes should therefore, not be used when devising the rules and the candidate keys. Instead, we should make use of the most informative attributes.
- Determine candidate rules for the matching ruleset based on the weights of each attribute. The weights can reveal certain combinations of attributes that can be used to decide if a mapping (matching rule) should exist between a record pair. This enables the automated creation of the matching ruleset.

The remainder of this thesis is structured as follows. Chapter 2 summarises the main principles of the comparison subspaces, attribute matching techniques, and the three approaches (Rule-based, Cocktail, and Probabilistic-based) to detect duplicates. Chapter 3 gives an overview of the system implemented in this project. Chapter 4 focuses on the detailed implementation of each component in the system. Chapter 5 describes the results of our experiments that have been conducted to compare between the Rule-based

approach, the Probabilistic-based approach, and our Cocktail approach. Finally, Chapter 6 summarises our findings.

Chapter 2

Theoretical Background

In this section, we present theoretical background on the important properties of duplicate record detection.

2.1 Comparison Subspaces

A searching method in record linkage involves the creation of comparison subspaces that reduce the number of candidate record comparison pairs to a feasible number without compromising the quality of the duplicate detection process. The reduction is imperative to improve the efficiency of the system particularly when operating on a large set of data. There are several methods [16, 18] that can be used to reduce the cost of record comparison. For this project, we utilised two of them, which are the traditional blocking technique, and the Sorted Neighbourhood method.

2.1.1 Blocking Technique

Blocking [17] is achieved by sorting the the database on one or combinations of a few attributes as the blocking key, then separating the records into mutually exclusive partitions based on their agreement on the blocking key. Therefore, each record will only be assigned to a block. For example, if the blocking key is the states in the United States, the dataset will be sorted and partitioned into blocks where the first block contains records with state 'AR', the second block contains records with state 'AK', and so on. Only records in the same block are considered for comparison.

First Name	Last Name	Address	ID	Key
Sal	Stolfo	123 First Street	45678987	STOSAL123FRST456
Sal	Stolpho	123 First Street	45688987	STOSAL123FRST456
Stolfo	Sal	123 First Street	45688987	SALSTO123FRST456

Figure 2.1: Sorting keys generated for the SN method

2.1.2 Sorted Neighbourhood Method

The Sorted Neighbourhood (SN) method by Hernandez et. al. [7] also sorts the records based on a sorting key. The key for each record is computed by extracting relevant attributes or portions of the attributes' values. Relevant attributes should have sufficient discriminating power in identifying records that are likely to be duplicates. Figure 2.1 shows how sorting keys might look like. The keys are a combination of sub-lengthed values of attributes in the order of last name, first name, address, and ID.

Since the records are sorted using the key generated, the attribute that first appears in the key selection has the most discriminating power, followed by the attributes that appear subsequently with decreasing discriminating power. Therefore, it is important that the first attribute is chosen with care.

To deal with large datasets, Hernandez and Stolfo employed a window of fixed size w, which moves sequentially over the dataset. Every new record that enters the window is compared with the previous $w-1$ records to find matching records (refer to Figure 2.2). Hernandez and Stolfo also suggested the use of multi-pass technique to sort the dataset not only once, but multiple times using different sorting key for each single-pass. The sliding window is applied throughout the whole dataset during each single-pass. This technique is to increase the possibility of duplicates falling into the same window. For example, the second and third persons in Table 2.1 refer to similar person. However, the first and last name of the third person have been transposed. Consequently, the third record might not end up in the same sliding window as the first and second records. Records not in the same window will not be considered as candidate record pairs. With the multi-pass technique, a second candidate key can be created from the combination of address, ID, and last name. Subsequently, we increase the chance for all three records falling into the same sliding window.

Transitive closure is then applied to merge the results from the multi-pass to detect relationships between duplicates that are not detected by the Equational Theory. For instance, the first and second records in Figure 2.1 are found to be similar during the first single-pass, whilst the second and third records are found to be similar during the second single-pass. Transitive equality will infer that the first and third records refer to the same person.

Meanwhile, it is proven in [7] that using a window with relatively small *w* over several independent passes returns better results than using a window with large *w* over a single pass.

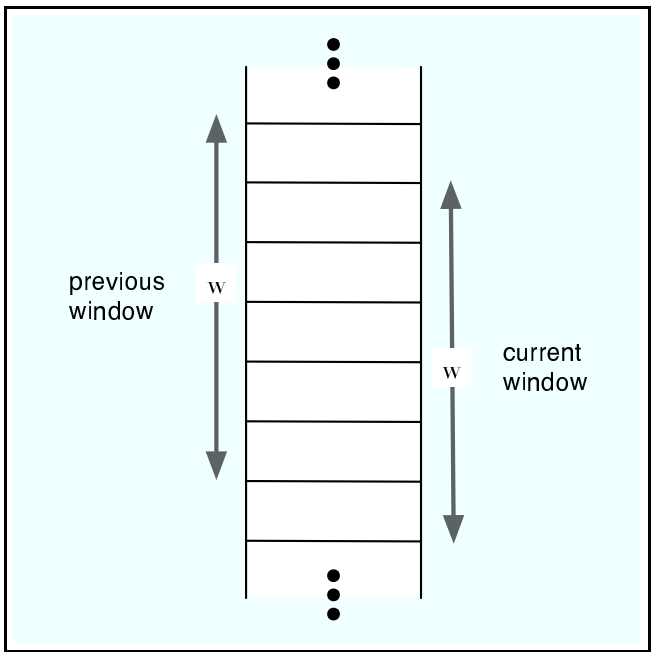

Figure 2.2: Sliding Window

2.2 Attribute Matching Techniques

Duplicate detection relies heavily on string comparison techniques to determine if two records are matching or non-matching. It is typical to compare for string similarity instead of equality because it is common that mismatches happen due to typographical error. Various string similarity techniques have been created [16, 19] and they can be categorised as character-based similarity metrics and token-based similarity metrics. In the following sections, we briefly discuss about the similarity metrics that are used in this project.

2.2.1 Character-Based Similarity Metrics

The character-based similarity metrics are designed to handle errors due to typography well. We cover one of them in this section.

2.2.1.1 Needleman and Wunsch

Needleman and Wunsch [20] is an edit distance metric that measures the amount of differences between two data strings. Similar to the Levenshtein distance metric, it calculates the number of operations needed to transform one string into the other. The operation can be an insertion, a deletion, or a substitution of a single character. Needleman and Wunsch extends over the Levenshtein distance metric by allowing for varying costs for different edit distance operations. For example, the cost of replacing '0' with 'O' is smaller than the cost of replacing 'a' with 'q'.

2.2.2 Token-Based Similarity Metrics

Character-based similarity metrics do not work well for situations where there are rearrangements of words for instance, "John Smith" versus "Smith John". The obvious similarity between the two strings failed to be captured by the character-based similarity metrics. Token-based similarity metrics compensate for this problem by breaking the strings into tokens of words. Jaccard is one such metrics.

2.2.2.1 Jaccard

The Jaccard [21] metric was initially developed to assess similarity between distributions of flora in different geographical areas. It uses word sets from the comparison strings to evaluate similarity. Each data string is represented as a Jaccard vector similarity function. The Jaccard between two data strings, X and Y, is represented as:

$$\frac{X * Y}{|X||Y| - (X * Y)} \tag{2.1}$$

where $(X * Y)$ is the inner product of X and Y, and $|X| = \sqrt{(X * X)}$, the euclidean norm of X.

2.3 Duplicate Records Detection

2.3.1 Probabilistic-Based Approach

In this section, we provide background on:

1. The Fellegi-Sunter model of record linkage.
2. The Expectation Maximization (EM) algorithm that computes the maximum likelihood estimates required by the Fellegi-Sunter model without training data.

2.3.1.1 Fellegi-Sunter Model of Record Linkage

Fellegi and Sunter [1] formalised a mathematical model based on ideas introduced by Newcombe [22]. The model aims to classify a record pair as matching or non-matching given the density function of each class. We explain how this works in the next paragraphs using the same notations provided in [1].

We represent the set of ordered record pairs (with a record drawn from each file A and B for each pair) as

$$AXB = \{(a,b); a \in A, b \in B\} \tag{2.2}$$

Each record pair is assigned to either class M or U. Record pairs belonging to the M class are identified as matching whilst record pairs belonging to the U class are identified as non-matching. Therefore, the set of pairs in (2.2) is the union of two disjoint sets

$$M = \{(a,b); a = b, a \in A, b \in B\} \tag{2.3}$$

and

$$U = \{(a,b); a \neq b, a \in A, b \in B\} \tag{2.4}$$

which we call the *matched* and *unmatched* sets respectively.

The records corresponding to members of A and B are denoted by $\alpha(a)$ and $\beta(b)$ respectively. We represent the record pairs as comparison vectors

$$\gamma[\alpha(a), \beta(b)] = \{\gamma^1[\alpha(a), \beta(b)], ..., \gamma^K[\alpha(a), \beta(b)]\} \tag{2.5}$$

where each of the $\gamma^i, i = 1, ..., K$ represents a specific comparison between the same i-th attribute of A and B. For example, γ^1 could denote the agreement or disagreement between the name of two persons. The degree of agreement/disagreement varies and could be such

that the name must be exactly the same and contain the word 'Brown', the name is almost similar, or the name disagrees.

Taking Γ as the comparison space that represents the set of all possible realizations of γ, a linkage rule L is defined as a mapping from Γ onto a set of random decision functions $D = \{d(\gamma)\}$ where

$$d(\gamma) = \{P(A_1|\gamma), P(A_2|\gamma), P(A_3|\gamma)\}; \gamma \in \Gamma \tag{2.6}$$

and

$$\Sigma_{i=1}^{3} P(A_i|\gamma) = 1$$

The three decisions are denoted as *link*(A_1), *non-link*(A_3), and *possible link*(A_2). The possible link class is introduced for ambiguous cases such as insufficient information. Rather than falsely classifying the record pair that falls under these cases as not linked, it is safer to classify it as a possible match for further examination. A record pair is considered linked if the probability that it is a link, $P(M|\gamma[\alpha(a), \beta(b)])$, is greater than the probability that it is a non-link, $P(U|\gamma[\alpha(a), \beta(b)])$. To solve this, we follow the Bayes decision rule for minimum error which introduces a likelihood ratio based on conditional probabilities of the record pair when it is known to be a link or non-link. These values can be computed using a training set of pre-labelled record pairs or using the EM algorithm (refer to Section 2.3.1.2).

The likelihood ratio is defined as

$$R = R[\gamma(a,b)] = m(\gamma)/u(\gamma) \tag{2.7}$$

where the conditional probability of $\gamma(a,b)$ if $(a,b) \in M$ is given by

$$m(\gamma) = P\{\gamma[\alpha(a), \beta(b)] | (a,b) \in M\} = \sum_{(a,b)\in M} P\{\gamma[\alpha(a), \beta(b)]\} \cdot P[(a,b)|M] \tag{2.8}$$

and the conditional probability of $\gamma(a,b)$ if $(a,b) \in U$ is given by

$$u(\gamma) = P\{\gamma[\alpha(a), \beta(b)] | (a,b) \in U\} = \sum_{(a,b)\in U} P\{\gamma[\alpha(a), \beta(b)]\} \cdot P[(a,b)|U] \tag{2.9}$$

The model by Fellegi and Sunter also includes errors associated with the linkage rule. The first type of error occurs when an unmatched pair of records is classified as a link, which is also the error of making decision A_1. This error has the probability of

$$\mu = P(A_1|U) = \sum_{\gamma\in\Gamma} u(\gamma) \cdot P(A_1|\gamma) \tag{2.10}$$

The second type of error is the error of decision A_3 that occurs when a matched pair of records is classified as a non-link. This error has the probability of

$$\lambda = P(A_3|M) = \sum_{\gamma \in \Gamma} m(\gamma) \cdot P(A_3|\gamma) \tag{2.11}$$

Fellegi and Sunter defined an optimal linkage rule L_0 with A_1, A_2, and A_3 according to the theorem [23]:

Theorem: Let L' be a linkage rule associated with decisions A'_1, A'_2, and A'_3 such that $P(A'_3|M) = P(A_3|M)$ and $P(A'_1|U) = P(A_1|U)$. Then L_0 is optimal in that $P(A_2|U) \leq P(A'_2|U)$ and $P(A_2|M) \leq P(A'_2|M)$.

This property is desirable to minimize the probability of making non-conclusive decision, A_2. In other words, the rule minimizes the probability of failing to make a conclusive decision subject to the fixed levels of error in (2.10) and (2.11).

From (2.7), the upper bound and lower bound thresholds are defined as

$$T_\mu = \frac{m(\gamma_n)}{u(\gamma_n)} \tag{2.12}$$

and

$$T_\lambda = \frac{m(\gamma_{n'})}{u(\gamma_{n'})} \tag{2.13}$$

With this, the pairs of error levels (μ, λ) corresponding to T_μ and T_λ are given by

$$\mu = \sum_{\gamma \in \Gamma_\mu} u(\gamma) \tag{2.14}$$

$$\lambda = \sum_{\gamma \in \Gamma_\lambda} u(\gamma) \tag{2.15}$$

where

$$\Gamma_\mu = \{\gamma : T_\mu \leq m(\gamma)/u(\gamma)\} \tag{2.16}$$

$$\Gamma_\lambda = \{\gamma : m(\gamma)/u(\gamma) \leq T_\lambda\} \tag{2.17}$$

2.3.1.2 EM Algorithm

The EM algorithm [17, 24] is a means of obtaining maximum likelihood estimates for incomplete data. It is a good alternative in situations where we cannot get hold of a subset of training data to compute the conditional probabilities as given in (2.8) and (2.9). Also, the dataset itself holds plenty of information that we can harness from.

The EM algorithm generates the parameter set $\Phi = (m, u, p)$ via iterative computation of the E-step (Expectation) and M-step (Minimization) on an incomplete data set. The steps are:

1. Give an initial estimated values of Φ. These values can be simply guessed as the algorithm is not particularly sensitive to the starting values.

2. Compute the E-step using the values of Φ.

3. Compute the M-step to re-estimate the values of Φ based on the values from Step 2.

4. Repeat Step 2 and Step 3 until the convergence of the values of Φ.

To estimate m_i, the EM algorithm needs to consider matching record pairs. One way that is often used is by using the blocking technique to group similar records into their respective blocks. Records pairs formed from within the same block are used to compute the comparison vector γ.

Jaro [17] used a binary model for the comparison vector γ such that $\gamma_i^j = 1$ if attribute i agrees for record pair j, and $\gamma_i^j = 0$ if attribute i disagrees for record pair j, for $i = 1, \ldots, n$ attributes and $j = 1, \ldots, N$ record pairs. Hence, the m_i and u_i probabilities can be defined as

$$m_i = P\{\gamma_i^j = 1 | r_j \in M\} \tag{2.18}$$

and

$$u_i = P\{\gamma_i^j = 1 | r_j \in U\} \tag{2.19}$$

The notation p is defined as the proportion of matched pairs where

$$p = \frac{|M|}{|M \cup U|} \tag{2.20}$$

Assuming an independence model, the conditional probabilities given in (2.8) and (2.9) are computed as

$$P(\gamma^j | M) = \prod_{i=1}^{n} m_i^{\gamma_i^j} (1 - m_i)^{1 - \gamma_i^j} \tag{2.21}$$

$$P(\gamma^j | U) = \prod_{i=1}^{n} u_i^{\gamma_i^j} (1 - u_i)^{1 - \gamma_i^j} \tag{2.22}$$

To compute the E-Step, let x be the complete data vector equal to $\langle \gamma, g \rangle$, where $g_j = (1,0)$ iff $r_j \in M$ and $g_j = (0,1)$ iff $r_j \in U$. The complete data log-likelihood is [25]:

$$\ln f(x|\Phi) = \sum_{j=1}^{N} g_j \cdot (\ln P\{\gamma^j|M\}, \ln P\{\gamma^j|U\})^T + \sum_{j=1}^{N} g_j \cdot (\ln p, \ln(1-p))^T \tag{2.23}$$

Now, replace g_j with $(g_m(\gamma^j), g_u(\gamma^j))$ where

$$g_m(\gamma^j) = \frac{p \prod_{i=1}^{n} m_i^{\gamma_i^j} (1-m_i)^{1-\gamma_i^j}}{p \prod_{i=1}^{n} m_i^{\gamma_i^j} (1-m_i)^{1-\gamma_i^j} + (1-p) \prod_{i=1}^{n} u_i^{\gamma_i^j} (1-u_i)^{1-\gamma_i^j}} \tag{2.24}$$

$$g_u(\gamma^j) = \frac{(1-p) \prod_{i=1}^{n} u_i^{\gamma_i^j} (1-u_i)^{1-\gamma_i^j}}{p \prod_{i=1}^{n} m_i^{\gamma_i^j} (1-m_i)^{1-\gamma_i^j} + (1-p) \prod_{i=1}^{n} u_i^{\gamma_i^j} (1-u_i)^{1-\gamma_i^j}} \tag{2.25}$$

The values of $(g_m(\gamma^j), g_u(\gamma^j))$ are used in the following M-Step, which yields the following equations for the parameter set Φ after setting the partial derivatives for each of the three maximization problems to zero.

$$m_i = \frac{\sum_{j=1}^{N} \gamma_i^j \cdot g_m(\gamma^j)}{\sum_{N}^{j=1} g_m(\gamma^j)} \tag{2.26}$$

$$u_i = \frac{\sum_{j=1}^{N} \gamma_i^j \cdot g_u(\gamma^j)}{\sum_{N}^{j=1} g_u(\gamma^j)} \tag{2.27}$$

$$p = \frac{\sum_{j=1}^{N} g_m(\gamma^j)}{N} \tag{2.28}$$

2.3.1.3 Alternative Computation of u_i

Although the EM algorithm can derive both the values of m_i and u_i, the latter may not be accurate due to the biased situation in which it is derived from. It is suggested in [17] that an alternative method can be used to estimate u_i. Estimation of u_i is simplified by the fact that the cardinality of non-matching record pairs in a dataset is much greater than that of matching record pairs. Therefore, u_i can be obtained by considering the probability of chance agreement of the attribute i. Usually, this is done by considering a sample rather than all the record pairs in the dataset.

2.3.1.4 Composite Weights

The weight of each attribute of a record pair is computed based on the values in (2.26) and (2.27). If attribute i of the record pair matches, the weight of that attribute is given by:

$$w_i = \log_2 \left(\frac{m_i}{u_i} \right) \tag{2.29}$$

If attribute i disagrees, then the weight is

$$w_i = \log_2 \left(\frac{1 - m_i}{1 - u_i} \right) \tag{2.30}$$

We obtain the score of each record pair by summing up all the weights of attribute i. Attributes that agree make a positive contribution to this sum whilst attributes that disgree make a negative contribution to this sum. Hence, based on the optimal decision rule for record linkage by Fellegi and Sunter, if the composite weight of a record pair is above threshold value T_μ (2.12), the record pair is classified as a match. If the composite weight is below threshold value T_λ (2.13), the record pair is classified as a non-match. If the composite weight falls between these two values, the record pair is regarded as non-conclusive.

2.3.2 Rule-Based Approach

The Rule-based approach is the use of rules to define whether two records are similar or not. It differs from the Probabilistic-based approach in the sense that each attribute is given either a weight of one or zero whilst each attribute in the Probabilistic-based approach is assigned a weight.

The use of the Rule-based approach dates back to 1989, where Wang and Madnick [26] proposed the use of heuristic rules developed by experts to infer additional information about the data instances to be matched. For example, an expert might define rules such as:

```
IF age < 21
    THEN student_status = undergraduate
    ELSE student_status = graduate

IF course_id = 564 AND course_id = 579
    THEN student_major = MIS
```

Table R

name	**cuisine**	**street**
TwinCities	Chinese	Wash. Ave.
TwinCities	Indian	Univ. Ave.

Table S

name	**speciality**	**city**
TwinCities	Mughalai	St. Paul

Figure 2.3: ILFD Example

Wang and Madnick hoped to used such rules to cluster records that represent the same real-world entity. However, since the knowledge is heuristically derived, the matching results produced may not be correct.

Lim et. al. [8] further developed this idea and arrived at ILFD (Instance Level Functional Dependency) that relies on functional dependencies rather than heuristic rules. The ILFDs are used to derive an extended key, which is a union of keys or attributes from the dataset. As an example, consider the two tables in Figure 2.3. Both Table R and Table S do not share any common keys because ”TwinCities” in Table S can refer to either one of the records in Table R. To solve this, we can assert in the following IFLD that the values of attribute *speciality* can infer the values of attribute *cuisine*. Hence, Table R and Table S can be matched using the extended key *(name, cuisine)* and the corresponding extended key equivalence rule, which states that two records are similar when their *name* and *cuisine* values are the same.

```
ILFD:
    (e2.speciality = Mughalai) -> (e2.cuisine = Indian)

Extended Key Equivalence Rule:
    (e1.name = e2.name) AND (e1.cuisine = e2.cuisine)
        -> (e1 and e2 are the same)
```

The Equational Theory suggested by Hernandez and Stolfo [7] is a more recent work that dictates the logic of domain equivalence. An example rule that exemplifies one axiom of the Equational Theory for an employee database is:

```
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
```

The implementation of *differ slightly* in the above rule is based upon the distance-based technique. A string-matching algorithm is applied on the first names of two records to determine the typographical gap. The gap is then compared with a certain threshold to decide if the two records are matching or non-matching. A poor choice of threshold value would result in false positives or false negatives. For instance, if we set a strict threshold value, we would have missed a number of duplicates. On the other hand, relaxing the value of threshold also means that non-duplicates would be misclassified as matching records. Therefore, a good matching declarative rule very much depends on the selection of distance functions and a proper threshold. The threshold value is normally obtained through experimental evaluation.

2.3.3 Cocktail Approach

The weaknesses of the rule-based approach lie in the exhaustive manual tuning of rules which may require a deep understanding of the data domain and the behaviour of the data itself. It is difficult to design a set of matching rules that best describe the datasets without spending hours analysing the data. Desirably, we want to be able to confidently create rules without examining the data for two reasons:

1. We may be dealing with private databases where data cannot be revealed.
2. Two datasets with similar domain may behave drastically different, which means that the rules manually crafted for the first dataset may not be applicable for the second dataset.

Consequently, we want to have an indication about the dataset without observing the data, but by getting some statistics to better understand the behaviour of the data. The Probabilistic-based approach allows us to obtain these statistics.

The Probabilistic-based approach has a way of telling us which attributes make the most contributions to an accurate mapping and which attributes do not in the form of matching weight score (2.29). For our first contribution, we suggest using this knowledge to:

1. Design rules that only include contributions from significant attributes.
2. Design candidate keys leveraging on significant attributes for the multi-pass Sorted Neighbourhood method.

For the first item, attributes that are not as significant can be totally removed from the rules since their presence in the matching rules are non-effective. An attribute may not be considered important if the matching weight score is relatively low compared to the matching weight scores of the rest of the attributes in the dataset. A low weight score means that the probability that a record pair matches (2.26) is low or the probability that a record pair do not match (2.27) is high for a particular attribute i. Since m_i and u_i are made from observing the data itself (using the EM algorithm), we can deduce that m_i is low due to missing data, frequent mistakes of entering data into i or the information obtained for i is simply not correct most of the time. Meanwhile, if u_i is high, this is probably because the data are very common throughout the whole dataset for i. For example, the attribute *gender* which only has either F (for female) or M (for male) will obviously result in a high u_i, rendering the attribute useless to consider for matching a record pair. For the second item, insignificant attributes will not increase the chance of matching records falling into the same sliding window, especially attributes with very low value of m_i.

Our second contribution is making use of a subset of the EM algorithm again to automatically create a complete new set of rules. The vector patterns computed, γ, holds plenty of information about which combinations of attributes appear the most frequent as matching and non-matching pairs. By comparing the frequency counts of the same vector patterns that were computed to calculate m_i in Section 2.3.1.2 and u_i in Section 2.3.1.3, we can have an idea of which combination of attributes (a rule) that allows us to detect more duplicates without sacrificing too much on the precision. We refer the reader to Section 4.4.3 for some simple methods that we used to choose the rules.

Chapter 3

System Overview

3.1 System Architecture

The system we have created (refer to Figure 3.1) for this project consists of the following working components:

1. The Database (MySQL)
2. Candidate Key Creator
3. Comparison Space Creator
    (a) Sliding Window Component
    (b) Blocks Component
4. Records Pairing Creator
    (a) Sliding Window Component
    (b) Blocks Component
5. Matchers
    (a) Rule-based Matcher and Rule-based Components
    (b) Probabilistic-based Matcher and Probabilistic-based Components
    (c) Cocktail Matcher
6. Classifier

7. Evaluator

The MySQL database contains the Restaurant, Cora, and DBGen datasets. For each dataset that is evaluated, the entire dataset is queried and loaded into memory for the Candidate Key Creator to generate the candidate keys for the records. The candidate keys are used mainly to sort the dataset for the Comparison Space Creator.

The Comparison Space Creator consists of two components that employ the blocking technique and sliding window technique respectively. Each time a block or sliding window is created from the data in memory (sorted according to candidate keys), all the records in the block or sliding window are passed to the Records Pairing Creator. The Records Pairing Creator then pairs up the records and feeds them pair by pair to the Matcher. The Matcher is comprised of a Rule-based Matcher that employs the Rule-based approach, a Probabilistic-based Matcher that employs the Probabilistic-based approach, and the Cocktail Matcher that employs the Cocktail approach, an integration between the Probabilistic-based and Rule-based approaches.

Upon receiving a record pair from the Records Pairing Creator A, the Rule-based Matcher proceeds to compute the similarity scores for each attribute in the record pair. The scores are sent to the Equational Theory rules to check if they satisfy the threshold. If they do, the record pair is considered matching and is sent to the Classifier.

The Probabilistic-based Matcher requires the dataset to be evaluated twice: (1) To estimate the threshold values given by Equation (2.12) and Equation (2.13), and (2) to determine if a record pair is matching or non-matching. To estimate the threshold values, the Probabilistic-based Matcher computes the comparison vectors for each record pair it receives from Records Pairing Creator B. All record pairs in all blocks that have been converted into comparison vectors are forwarded to the EM Algorithm to estimate m_i (refer to Equation (2.18)). Meanwhile, the values of u_i, given by Equation (2.19), are estimated by the Alternative Method component. The values of m_i and u_i are sent to the Threshold Generator to compute the threshold values. For the second part, which is where the real matching process begins, we used the sliding window method to obtain record pairs. Each generated record pair from Records Pairing Creator A is fed into the Probabilistic-based Matcher to compute the composite weight of the record pair. The composite weight is compared with the threshold values. If the weight is above the upper bound threshold value, the record pair is regarded as matching and is passed to the Classifier.

Similar to the Probabilistic-based Matcher, the Cocktail Matcher also needs to run through the dataset twice. The first part is similar to the Probabilistic-based Matcher where the Cocktail Matcher converts each record pair from Records Pairing Creator B to vector patterns. Then, the values of m_i and u_i are computed. These values are used to generate a matching ruleset by the Equational Theory Rules component. The second part is similar to the Rule-based Matcher where the Cocktail Matcher computes similarity scores for each attributes of the record pair it receives from Records Pairing Creator A. The scores are compared with the threshold of the Equational Theory ruleset to check if the record pair is matching or non-matching. If it is found to be matching, the record pair is forwarded to the Classifier.

The Classifier is responsible for the following:

1. Determine records that should be grouped together on the basis that all records in the same group are duplicates of each other.

2. Determine if previous distinctly-grouped records on the basis that all records in group A do not match all records in group B stays true after the next matching process.

3. Compute transitive closure over several independent runs.

Grouped duplicates are then forwarded to the Evaluator to assess the performance of the matching system.

The Evaluator assesses the performance of the three approaches using the following four performance metrics.

1. Recall, which is the fraction of duplicates correctly classified over the total number of duplicates in the dataset. The formula is:

$$Recall = \frac{truepositives}{truepositives + falsenegatives} \tag{3.1}$$

2. Precision, which is the fraction of correct duplicates over the total number of record pairs classified as duplicates by the system. The formula is:

$$Precision = \frac{truepositives}{truepositives + falsepositives} \tag{3.2}$$

3. F-Score or F-measure, which is the harmonic mean of the precision and recall values and is give by:

$$F - Score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{3.3}$$

4. Time complexity, which measures how well the system scales in terms of time.

Bilenko et. al. [27] advocates precision-recall curves as the most informative evaluation methodology. It is most useful when assessing the system. For example, if recall is the more important attribute, the approach with higher recall although sacrificing on precision, would be chosen.

3.2 Technologies

The system is built using the following technologies.

3.2.1 Java Development Kit (JDK)

The core of the system is built using the Java Platform (Standard Edition 6). Java is chosen because of its cross-platform capabilities. Since the system is tested on a MacBook running Tiger version 10.4.11 and Apple does not release Java SE 6 for Tiger, we have opted to use SoyLatte [1], which is FreeBSD port of Sun's Java 6 JDK. The version of SoyLatte used in this project is 32-bit JDK 6 1.0.2 [2].

3.2.2 MySQL Database System

MySQL [3] is the world's most popular open source database, as quoted from its website [4]. Due to its free usage and fast performance, it is favoured as the relational database management system (RDBMS) of choice for this project.

3.2.3 Java Database Connectivity (JDBC)

The Java Database Connectivity (JDBC) API provides the functionality to access the database system from within the Java program. It provides functionalities to access and

[1] http://landonf.bikemonkey.org/static/soylatte/
[2] http://hg.bikemonkey.org/archive/javasrc_1_6_jrl_darwin/soylatte16-i386-1.0.2.tar.bz2
[3] http://www.mysql.com/
[4] http://www.mysql.com/why-mysql/

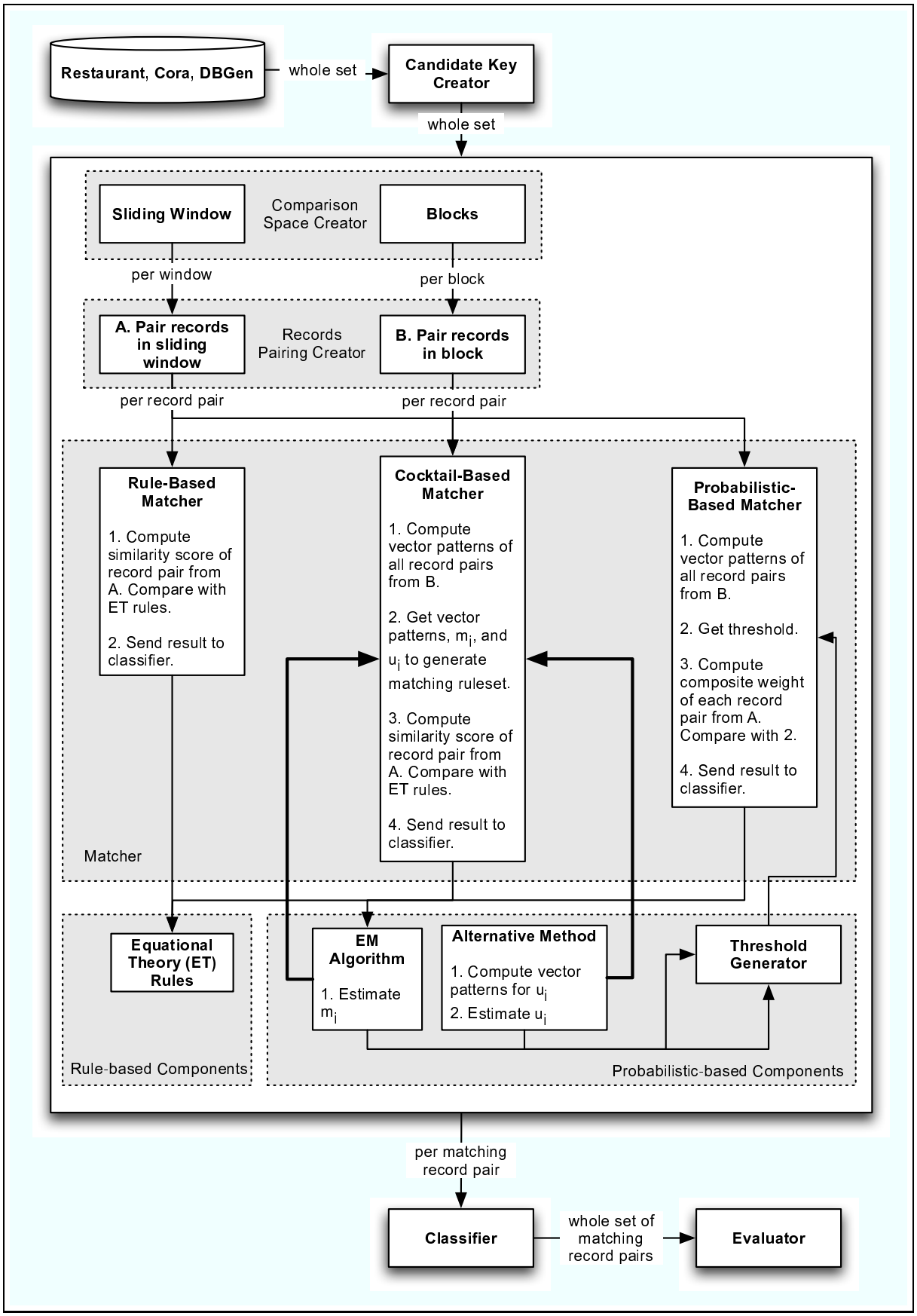

Figure 3.1: System Architecture

retrieves data from the database. The MySQL JDBC driver is downloadable from the MySQL website [5]. The version of the connector used for this project is 5.0.7.

3.2.4 Waikato Environment for Knowledge Analysis (Weka)

Weka [6] is a library of Java routines for machine learning tasks developed by the Machine Learning Project at the Department of Computer Science of The University of Waikato. It is freely available under the GNU General Public License and is fully implemented in Java. We use the Weka Java library for processing datasets stored in Attribute-Relation File Format (ARFF) [7] files, which are ASCII text files that describe a list of instances sharing a set of attributes. Two of the datasets we are using in this project are available in the ARFF format.

3.2.5 SimMetrics

SimMetrics [8] is an open source Java library of distance metrics. It contains programs for a variety of edit distance and token-based metrics such as Smith-Waterman, Jaro, Needleman-Wunsch, Jaccard, TFIDF, Levenstein, and Q-Gram distance. It takes in a pair of string data and return the distance result in the form of floating-point based number (0.0 - 1.0).

[5] http://dev.mysql.com/downloads/connector/j/
[6] http://www.cs.waikato.ac.nz/ ml/weka/
[7] http://www.cs.waikato.ac.nz/ ml/weka/arff.html
[8] http://www.dcs.shef.ac.uk/ sam/simmetrics.html

Chapter 4

Implementation Details

4.1 Candidate Key Creator

When transferring the datasets to the database, we added an attribute *candKey* to each dataset to store the candidate keys of the records. This is to enable us to query the whole dataset only once and avoiding queries such as a join. The creation of candidate keys differ for both the blocking technique and the SN method.

We used the blocking technique only to compute the EM algorithm. As such, the SQL statement to create candidate keys for the blocks is slightly different from the SN method. The following example of SQL statement shows that we only create candidate keys for records with non-empty data for the first attribute in the candidate key. If we do not ignore the empty data, we would have obtained a very low u_i for *venue*, if there are only thirty records with non-empty *venue* from a dataset of one thousand records. However, since we want to evaluate the importance of attribute *venue*, it is important to evaluate 'venue' when its value of m_i is high to compute the comparison vector that has *venue* as non-zero. Subsequently, we would be able to create a matching rule for *venue* when a record pair agrees on it.

```
UPDATE Restaurant SET candKey=UPPER(CONCAT(
SUBSTRING(venue, 1, 3), SUBSTRING(title, 1, 3)))
WHERE venue!=''
```

An example SQL statement to create candidate keys for sliding window is given below. Notice the difference with the previous SQL statement. Here, we create candidate keys for all records because we want the Matcher to evaluate the whole dataset.

```
UPDATE Restaurant SET candKey=UPPER(CONCAT(
SUBSTRING(venue, 1 3), SUBSTRING(title, 1, 3)))
```

4.2 Comparison Space Creator

We used the following SQL statement to order the database according to the candidate keys ascendingly, before loading the whole dataset into memory. Then, we applied either the blocking technique or the SN method.

```
SELECT * FROM tablename WHERE candKey!='' ORDER BY candKey ASC
```

4.2.1 Blocks Creator

For the blocking technique, all the blocks are non-overlapping windows. Also, they may not appear subsequently one after another, which means that there are records that do not belong to any blocks. In such cases, we ignore those records.

Algorithm 1 shows how the blocks are created. First, we retrieve the first record, A, and the second record, B, from the data in memory. Then, we compare the candidate key of record A (candKey A) with the candidate key of record B (candkey B). Next, we compute the similarity score between candKey A and candKey B using the Needleman and Wunsch metric, with threshold set to one. If the similarity score equals to one, record B is exactly the same as record A and we put it into the same block as A. We repeat the process with the record after record B. Hence, all subsequent records that satisfy the threshold are put into the same block as record A. If no subsequent records are found to satisfy the similarity test, we use the next record to re-identify a new block. Figure 4.1 shows an example of blocks identified by the algorithm. Once a new block is found, we retrieve the identifiers of all records in the block and pass the identifiers to Algorithm 3.

Since the EM algorithm relies on the blocking technique and we intend to limit the number of record pairs to estimate Equation (2.26) for large datasets, Algorithm 1 also includes a boolean (denoted as *quitSearch*) that keeps a check on the number of record pairs computed in Algorithm 3. Once the limit is reached, we stop looking for more record pairs.

Input: All sorted records from database (*data*)

Output: Identifiers of all records in a block (*ids*)

```
1  quitSearch ← false;
2  threshold ← 1;
3  while not end of data and !quitSearch do
4      A ← candidate key for the next record;
5      blockSize ← 1;
6      while not end of data do
7          B ← candidate key for the next record;
8          score ← similarity score between A and B based on Needleman and
           Wunsch similarity metric;
9          if score ≥ threshold then blockSize ← blockSize + 1;
10         else
11             go back to the previous record;
12             quit inner while loop;
13     if block_size > 1 then
14         go back to blockSize previous records;
15         ids ← identifiers of all records in block;
16         quitSearch ← pairRecords(ids);
17     else skip;
```

Algorithm 1: createBlocks()

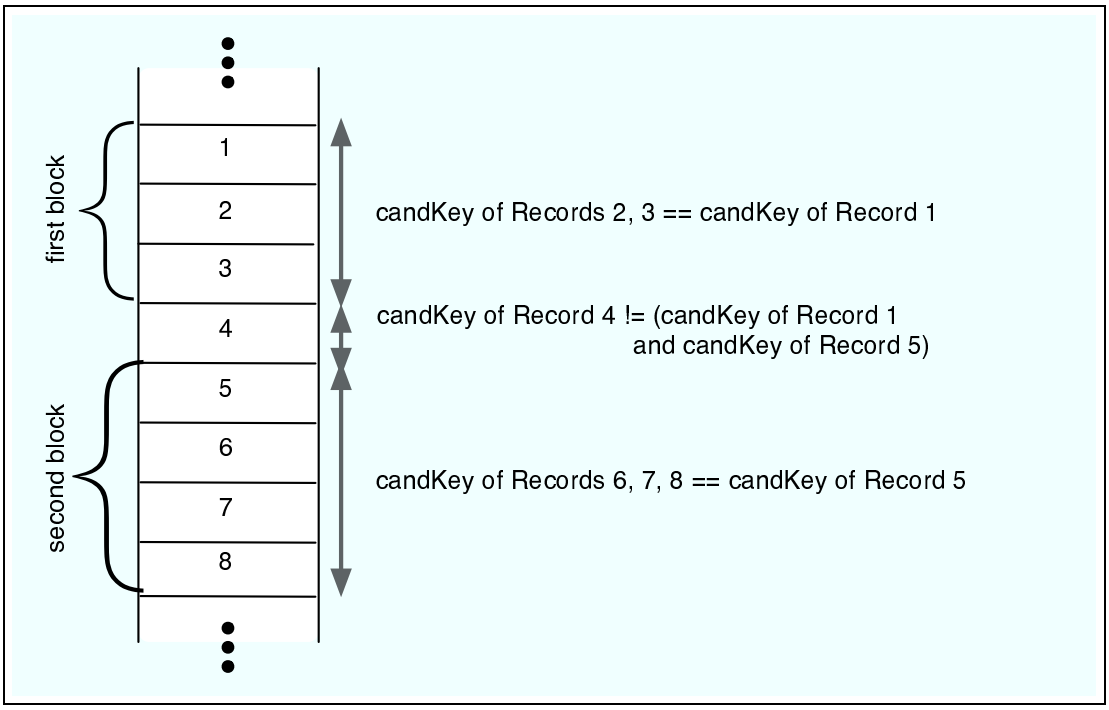

Figure 4.1: Creation of Blocks

4.2.2 Sliding Window Creator

The size of the sliding window is fixed, unlike the blocking technique. Only the very first window is like a block, hence records in the first window are paired up in the similar way as the blocking technique (line 3 to 4 of Algorithm 2). Each time the window slides down, we remove the first record and add the next record to the list of identifiers (line 6 and 7). These identifiers are then passed to Algorithm 3 for record pairing.

4.3 Records Pairing Creator

The way records are paired up in the blocking technique differs from the sliding window method. In this section, we provide the algorithms for these two methods. Record pairs created here are forwarded to the Matcher one by one to determine if they are matching or non-matching pair.

```
Input: All sorted records from database (data), database size (dbSize), sliding
       window size (winSize)
Output: Identifiers of all records in a block/sliding window (ids)
1  totalSlides ← dbSize − winSize;
2  for i ← 0 to totalSlides do
3      if i is 0 then
4          ids ← identifiers of all records in block from 0 to winSize − 1;
5      else
6          remove first record from sliding window;
7          add the next record to sliding window;
8          ids ← identifiers of all records in sliding window;
9      pairRecords(ids);
```

Algorithm 2: createSlidingWindow()

4.3.1 Pairing Records in Blocks

Using the list of identifiers of a block computed by Algorithm 1, we iterate through the list and let the identifier at each iteration be $t1$. The partner pair, $t2$, of $t1$ at each iteration is the rest of the records with positions higher than position $t1$ in the list. Line 1 to 13 of Algorithm 3 shows how record pairs in a block are created. The algorithm also returns a boolean *quitSearch* (line 12) to signify to Algorithm 1 that the limit of record pairs is reached.

4.3.2 Pairing Records in Sliding Window

Unlike the blocking technique, the only $t1$ for the SN method is the last record in the window. $t1$ is paired up with all the rest of the records in the window except itself. Line 14 to 18 of Algorithm 3 shows how record pairs in a sliding window are created.

Input: Identifiers of all records in a sliding window (*ids*), limit of record pairs in block (*limit*)

Output: Record pair $(t1, t2)$, *quitSearch*

```
1  if evaluating a block then
2      for id1 ∈ ids do
3          t1 ← id1;
4          for id2 ∈ ids where position of id2 > position of t1 in ids do
5              t2 ← id2;
6              compare(t1, t2);
7              increase totalPairs by 1;
8              if totalPairs > limit then
9                  quitId2 ← true;
10                 quit inner loop;
11         if quitId2 then
12             quitSearch ← true;
13             quit outer loop;
14 else if evaluating a sliding window then
15     t1 ← last identifier in ids;
16     for id ∈ ids except t1 do
17         t2 ← id;
18         compare(t1, t2);
```

Algorithm 3: pairRecords()

4.4 Matcher

4.4.1 Probabilistic-Based Matcher

The following steps show the overall picture of how the Probabilistic-based Matcher works:

1. Create candidate keys for the blocking technique (Section 4.1).
2. Create record pairs in blocks. This is done by Algorithm 1, which in turn calls Algorithm 3 to pair up the records in each blocks.
3. Compute the vector pattern of each record pair formed. Vector pattern is computed by comparing the similarity score of each attribute i of the record pair against a threshold value. If the score of i satisfies the threshold, the value of i is set to 1. Otherwise, it is set to 0. Hence, the vector pattern consists only of binary values. We store all the vector patterns γ^j and their respective frequency counts $f(\gamma^j)$ in a Collection object.
4. Estimate the values of m_i from the collection of vector patterns and frequency counts. Refer to Section 4.4.1.1 for more details.
5. Estimate the values of u_i. Refer to Section 4.4.1.2 for more details.
6. Find the threshold value using the computed m_i and u_i. Refer to Section 4.4.1.3 for more details.
7. Create candidate keys for the SN method (Section 4.1).
8. Create sliding window (Algorithm 2) and pair up records in the window (Algorithm 3).
9. Calculate composite weight of each record pair in the window. Refer to Section 4.4.1.4 for more details. Compare the composite weight with the threshold value to determine if the record pair is matching, non-matching, or is non-conclusive (refer to Algorithm 4).
10. Send record pairs found to be matching to the Classifier.
11. Repeat Step 7 to 10 for multi-pass SN method.

Input: record pair $(t1, t2)$, *upperBound*, *lowerBound*
Output: -
if *compute EM algorithm* **then**
    compute pattern for $(t1, t2)$;
else
    weight $\leftarrow$ `computeCompositeWeight(t1, t2)`;
    if *weight* $\geq$ *upperBound* **then** `classifier(t1, t2)`;
    else if *weight* $>$ *lowerBound* **then** add $(t1, t2)$ to reject region;
    else remove $(t1, t2)$ from reject region;

Algorithm 4: compare()

4.4.1.1 Estimating m_i

In practice, the binary model generates 2^n patterns of comparison vectors. Therefore, we created a Java Collection object to store the possible patterns γ^j, and frequency counts $f(\gamma^j)$ for each patterns. If a record pair generates a comparison vector in the Collection, a count of one is added to the frequency count of that particular pattern. Otherwise, we simply add the new pattern and a frequency count of one into the Collection. This process is repeated for all the record pairs in all blocks. After the last block, we run the EM algorithm based on the patterns and their frequency counts. The equations in (2.26), (2.27), and (2.28) are substituted as follows [17]:

$$m_i = \frac{\sum_{j=1}^{2^n} \gamma_i^j \cdot g_m(\gamma^j) f(\gamma^j)}{\sum_{j=1}^{2^n} g_m(\gamma^j) f(\gamma^j)} \tag{4.1}$$

$$u_i = \frac{\sum_{j=1}^{2^n} \gamma_i^j \cdot g_u(\gamma^j) f(\gamma^j)}{\sum_{j=1}^{2^n} g_u(\gamma^j) f(\gamma^j)} \tag{4.2}$$

$$p = \frac{\sum_{j=1}^{2^n} g_m(\gamma^j) f(\gamma^j)}{\sum_{j=1}^{2^n} f(\gamma^j)} \tag{4.3}$$

Algorithms 5, 6, and 7 show how we finally obtain a converged value of m_i from the iterations of E-Steps and M-Steps. Initial values for m_i, u_i, and p were 0.9, 0.1, and 0.5 respectively. These values were purely based on guesstimate since the algorithm is not particularly sensitive to starting values.

Input: total attributes of dataset (n)

Output: m_i, u_i

$m \leftarrow 0.9$;

$u \leftarrow 0.1$;

$p \leftarrow 0.5$;

$convergeValue \leftarrow 0.00001$;

for $i \leftarrow 1$ **to** n **do**

    $m_i \leftarrow m$;

    $u_i \leftarrow u$;

while m_i, u_i *and p are not converged to convergeValue* **do**

    `runEStep(`m_i, u_i, p`);`

    `runMStep(`$g_m(\gamma^j)$, $g_u(\gamma^j)$`);`

Algorithm 5: computeMI()

Input: A collection of vector patterns γ^j and their respective frequency counts $f(\gamma^j)$ (*collectionVecPattern*), total attributes of dataset (n), m_i, u_i, p

Output: $g_m(\gamma^j)$, and $g_u(\gamma^j)$

for $\gamma^j \in$ *collectionVecPattern for* $j \leftarrow 1$ **to** *size of collectionVecPattern* **do**

    for $i \leftarrow 1$ **to** n **do**

        $\gamma_i^j \leftarrow$ the i-th component in γ^j;

        $probMatching \leftarrow probMatching * m_i^{\gamma_i^j} * (1 - m_i)^{(1-\gamma_i^j)}$;

        $probNotMatchin \leftarrow probNotMatching * u_i^{\gamma_i^j} * (1 - u_i)^{(1-\gamma_i^j)}$;

    $g_m(\gamma^j) \leftarrow$

    $p * probMatching / (\mathrm{p} * probMatching + (1.0 - \mathrm{p}) * probNotMatching)$;

    $g_u(\gamma^j) \leftarrow ((1.0 - \mathrm{p}) * probNotMatching) / (\mathrm{p} * probMatching + (1.0 - \mathrm{p}) * probNotMatching)$;

Algorithm 6: runEStep()

Input: A collection of vector patterns γ^j and their respective frequency counts $f(\gamma^j)$ (*collectionVecPattern*), total attributes of dataset (n), total number of record pairs (N), $g_m(\gamma^j)$, $g_u(\gamma^j)$

Output: m_i, u_i, p

for *i from 1 to n* **do**
  for γ^j *and* $f(\gamma^j) \in$ *collectionVecPattern for* $j \leftarrow 1$ **to** *size of collectionVecPattern* **do**
    $\gamma_i^j \leftarrow$ the i-th component in γ^j;
    $m_i\_numerator \leftarrow m_i\_numerator + (\gamma_i^j * g_m(\gamma^j) * f(\gamma^j))$;
    $u_i\_numerator \leftarrow u_i\_numerator + (\gamma_i^j * g_u(\gamma^j) * f(\gamma^j))$;
    $m_i\_denominator \leftarrow m_i\_denominator + (g_m(\gamma^j) * f(\gamma^j))$;
    $u_i\_denominator \leftarrow u_i\_denominator + (g_u(\gamma^j) * f(\gamma^j))$;
  $m_i \leftarrow m_i\_numerator / m_i\_denominator$;
  $u_i \leftarrow u_i\_numerator / u_i\_denominator$;
  $p\_numerator \leftarrow m_i\_denominator$;
$p \leftarrow p\_numerator / N$;

Algorithm 7: runMStep()

4.4.1.2 Estimating u_i

The values of u_i from the EM algorithm were not used due to the biased situation they were derived from. We applied the alternative method that estimates u_i from a random sample of record pairs. First, N records (where N is equivalent to the number of record pairs used to estimate m_i in the EM algorithm) are randomly retrieved from the database using the following SQL statement:

```
SELECT * FROM tableName ORDER BY RAND() limit 0, N
```

Then, we randomly paired up N record pairs from the pool of N records. The number N is chosen purely by guesstimate. The larger the number of record pairs is, the more accurate the value of u_i is although at the expense of higher computational time. As usual, the comparison vector for each record pair is computed. Hence, we end up with 2^n possible patterns and their respective frequency counts. The estimation of u_i is the ratio of chance agreements and the number of record comparisons made. We provide the algorithm to estimate u_i in Algorithm 8.

4.4.1.3 Computing Threshold Values

As indicated in Section 2.3.1.1, the two upper and lower bound threshold values are governed by the probability of the two errors for an optimal record linkage rule. Therefore, the threshold values depend on the error level tolerance. In this project, we choose to exclude the tolerance level to ignore non-conclusive decision, A_2, completely.

We computed the thresholds according to the steps in [17]. The algorithm is given by Algorithm 9.

1. Compute the scores (composite weight) for each of the 2^n comparison vectors.
2. Order the comparison vectors based on the scores ascendingly.
3. Calculate the estimated true-positive (2.21) and true-negative (2.22) probabilities of each comparison vector.
4. To select the thresholds, the naive way used in this experiment is by choosing a middle index of ascendingly-sorted scores and setting it as an upper bound and lower bound threshold to avoid reject region. The middle index is chosen as the

Input: A collection of vector patterns γ^j and their respective frequency counts $f(\gamma^j)$ (*collectionVecPattern*), total attributes of dataset (n), number of records (N), database ($tableName$)

Output: u_i

```
1  c ← collection of vector patterns γ^j and their respective frequency counts f(γ^j);
2  w ← randomly selected N records from database tableName using SQL;
3  totalPairs ← N;
4  for x ← 1 to totalPairs do
5      t1 ← record randomly retrieved from w;
6      t2 ← record randomly retrieved from w;
7      γ^j ← computed pattern for record pair (t1, t2);
8      if γ^j exists in c then
9          add 1 to the frequency counts f(γ^j) of γ^j;
10     else
11         insert γ^j and its frequency count of 1 to c;
12 for i ← 1 to n do
13     for γ^j and f(γ^j) ∈ collectionVecPattern for j ← 1 to size of
       collectionVecPattern do
14         γ_i^j ← the i-th component in γ^j;
15         if γ_i^j is 1 then
16             add f(γ^j) to counter;
17         u_i ← counter/totalPairs;
```

Algorithm 8: computeUI()

index where the estimated true-positive probability is higher than the estimated true-negative probability. An alternative way is by choosing an upper bound threshold defined as the maximum weight for a non-match decision where the sum of $P(\bullet|M)$ does not exceed the desired probability that a matched pair should be classified as unmatched. Meanwhile, the lower bound threshold is defined as the minimum weight for a match decision where 1 minus the sum of $P(\bullet|U)$ does not exceed the desired probability that an unmatched pair should be classified as matched. Weights between the two values are non-conclusive cases.

4.4.1.4 Computing Composite Weight

To determine if a record pair is matching or non-matching, the composite weight of each record pair in the sliding window is computed. The formula to compute the composite weight is similar to the formula used to calculate the composite weight of vector pattern. Algorithm 10 shows the calculation of the composite weight of a record pair. The weight is returned to the Probabilistic-based Matcher for comparison with the threshold score.

4.4.2 Rule-Based Matcher

The following steps show the overall picture of how the Rule-based Matcher works:

1. Create Equational Theory rules and set a threshold value for similarity score.
2. Create candidate keys for the SN method (Section 4.1).
3. Create sliding window (Algorithm 2) and pair up records in the window (Algorithm 3).
4. Calculate similarity score for each attribute *i* of each record pair in the window. Compare the score of *i* against the Equational Theory rules and threshold value to determine if the record pair is matching or non-matching (refer to Algorithm 11).
5. Send record pairs found to be matching to the Classifier.
6. Repeat Step 2 to 5 for multi-pass SN method.

Input: A collection of vector patterns γ^j and their respective frequency counts $f(\gamma^j)$ (*collectionVecPattern*), total attributes of dataset (n), m_i, u_i

Output: *upperBound*, *lowerBound*

for $\gamma^j \in$ *collectionVecPattern for* $j \leftarrow 1$ **to** *size of collectionVecPattern* **do**
    for $i \leftarrow 1$ **to** n **do**
        $\gamma_i^j \leftarrow$ the i-th component in γ^j;
        $probMatching_i \leftarrow probMatching_i * m_i^{\gamma_i^j} * (1-m_i)^{(1-\gamma_i^j)}$;
        $probNotMatching_i \leftarrow probNotMatching_i * u_i^{\gamma_i^j} * (1-u_i)^{(1-\gamma_i^j)}$;
        if γ_i^j *is 1* **then**
            $score_i \leftarrow score_i + \log_2(m_i/u_i)$;
        else
            $score_i \leftarrow score_i + \log_2((1-m_i)/(1-u_i))$;
sort $score_i$ ascendingly;
get the middle index i of sorted score;
while $probMatching_i > probNotMatching_i$ *at middle index* i **do**
    decrease i by 1;
    $middleScore \leftarrow$ sorted score at middle index i;
while $probMatching_i < probNotMatching_i$ *at middle index* i **do**
    increase i by 1;
    $middleScore \leftarrow$ sorted score at middle index i;
$upperBound \leftarrow middleScore$;
$lowerBound \leftarrow middleScore$;

Algorithm 9: computeEMThreshold()

Input: total attributes of dataset (n), vector pattern of a record pair (*pattern*), m_i, u_i, record pair $(t1, t2)$

Output: composite score of record pair (*score*)

```
1 for i ← 1 to n do
2     pattern_i ← the i-th component in pattern;
3     if pattern_i is 1 then
4         score_i ← score_i + log2(m_i/u_i);
5     else
6         score_i ← score_i + log2((1 − m_i)/(1 − u_i));
7 return score;
```

Algorithm 10: computeCompositeWeight()

Input: record pair $(t1, t2)$

Output: matching record pair $(t1, t2)$

```
1 compute similarity scores of each attribute in (t1, t2);
2 pass similarity scores to Equational Theory rules of dataset;
3 if (t1, t2) is found to be matching then
4     classifier(t1, t2);
```

Algorithm 11: compare()

4.4.3 Cocktail Matcher

The Cocktail matcher is an integration between the Rule-based and Probabilistic-based matchers. The following steps show the overall picture of how the Cocktail matcher works:

1. Similar to Step 1 to Step 5 of the Probabilistic-based Matcher in Section 4.4.1.
2. Use information from Step 3 to Step 5 of the Probabilistic-based Matcher to compute the weight of each attribute *i* when *i* agrees, and to automatically derive matching rules.
3. Repeat Step 1 and 2 with different blocking keys to derive a whole set of Equational Theory rules. Set a similarity threshold for the rules.
4. Similar to Step 2 to Step 6 of the Rule-based Matcher in Section 4.4.2.

Similar to the Probabilistic matcher, the first step is grouping similar records into their respective blocks, then pair up the records in the same blocks. All the vector patterns and their respective frequencies, computed in Algorithm 4, are passed to Algorithm 5 to estimate m_i. We run Algorithm 8 next to obtain an estimate of u_i. Both m_i and u_i are used to calculate the weight of each attributes when the attribute agrees. From these weights, we can assert whether a particular attribute is significant or not significant most of the time. As our first contribution, we claim that if the attribute has a very low weight compared to the rest of the attributes, it does not give any contribution to the matching rules. Consequently, we do not consider the attributes when designing the matching rules. Also, we do not consider attributes especially with a low value of m_i as an element in the candidate key.

For our second contribution, we created a new set of rules automatically from the dataset itself by leveraging the EM algorithm again. This time, we make use of the computed vector patterns γ^j and their respective frequency counts. By observing the vector patterns and the frequency counts of the vector patterns given that it is matching and non-matching, we should be able to choose the best rules that give us a high recall without sacrificing too much on precision. We illustrate how this works in the subsequent paragraphs.

Let γ^j be the vector pattern of a record pair j assuming that it is matching since the records were taken from the same block. Let a^{γ^j} be the frequency count of γ^j given that it

is matching and b^{γ^j} be the frequency count of γ^j given that it is non-matching. We assume that the vector pattern is a good matching rule if a^{γ^j} is sufficiently much higher than b^{γ^j}. A simple and naive way to compute this is by taking the relative frequency of $a^{\gamma^j} - b^{\gamma^j}$ over the total number of vector patterns, N. We deducted b^{γ^j} from a^{γ^j} because b^{γ^j} indicates the number of patterns similar to a^{γ^j} when it is in fact non-matching.

However, it is unlikely that $\frac{a^{\gamma^j}-b^{\gamma^j}}{N} = \frac{10-0}{100} = 0.1$ for a dataset of 100 matching record pairs is similar to $\frac{100-0}{1000} = 0.1$ for a dataset of 1000 matching pairs because we would consider $\frac{40-0}{1000}$ good enough to indicate that this vector pattern has the potential to be a good matching rule. One way to solve this problem is by using the similar idea in sublinear tf (term frequency) scaling [28] where the logarithm of the term frequency is considered instead of the full weight of the term frequency. Similarly, we can apply logarithm to $a^{\gamma^j} - b^{\gamma^j}$, as well as to N to lower the full impact of the total weight. The formula we used to calculate the relative impact of a vector pattern is

$$\frac{\log\left(a^{\gamma^j} - b^{\gamma^j}\right)}{\log N} \geq t1 \tag{4.4}$$

We illustrate the decision rule with two examples. Supposed $t1$ is set to 0.5. For a dataset of hundred record pairs that we match after applying the blocking technique, if $a^{\gamma^j} - b^{\gamma^j} = 10$, Equation (4.4) will be satisfied because $\frac{\log 10}{\log 100}$ equals to 0.5. Meanwhile, for a dataset of one thousand record pairs, we would need at least 32 ($a^{\gamma^j} - b^{\gamma^j} = 32$) matching pairs from the EM algorithm such that $\frac{\log 31}{\log 1000} = 0.5017$ meets the threshold. This is a better approach than requiring the second example to have 100 ($a^{\gamma^j} - b^{\gamma^j} = 100$) matching pairs so that $100/1000$ satisfies the same threshold as the first example at $10/100$.

The decision rule above is not quite enough to classify a vector pattern as a good matching rule. For example, we might get a frequency count a^{γ^j} of 868 for a vector pattern given that it is classified as matching record pair and a frequency count b^{γ^j} of 169 given that it is non-matching pair from a total record pairs of 1000. $\frac{\log(868-169)}{\log 1000}$ gives us 0.9482 which is probably high enough to satisfy the first decision rule. However, 169 non-matching pairs from the EM algorithm is hardly a good indicator that the vector pattern is suitable as a matching rule because it might results in many false positives. Consequently, we provided a second decision rule that dictates the maximum allowable b^{γ^j}. This can be determined easily by using the following formula.

$$\frac{b^{\gamma^j}}{N} \leq t2 \tag{4.5}$$

Therefore, for $t2$ at 0.02, the maximum allowable b^{γ^j} is two for a dataset with one hundred record pairs, and twenty for a dataset with one thousand record pairs. With these two decision rules, vector patterns that measure up to the relative weight of at least $t1$ and relative error of at most $t2$ are considered suitable as matching rules.

4.4.3.1 Automatic Extraction of Matching Rules

As described in the previous paragraph, vector patterns that satisfy the two decision rules, Equation (4.4) and Equation (4.5), are considered as candidates for the matching rules. We still need to filter through the list of vector patterns to select the best few. There are two things that we need to consider. First, we strip off the non-significant attributes from the vector patterns. Second, we do not have to consider all the vector patterns because a few of them may already describe the rest. For example, we do not need the combination of '*similar_name && similar_address && similar_city*' when '*similar_name && similar_city*' also satisfies the decision rules because any record pairs that satisfy the former rule also satisfy the later rule. The former rule is considered redundant. Algorithm 12 describes the selection of the best matching rules.

To make a selection of vector patterns suitable to be matching rules, first, each vector pattern is compared against the two decision rules. If the vector pattern satisfies the decision rules, we modify all the insignificant attributes to '0's. Next, we calculate the total number of fields with '1's, n_k, that each candidate vector patterns has. Vector patterns with similar n_k, where k is the total number of attributes, are grouped together. Then, we iterate through all the groups starting from the group with the lowest number of n_k, to filter out redundant vector patterns.

We illustrate how the filter works in the following example. Let's say the following vector patterns are candidates for matching rules because they have been found to satisfy the decision rules (grouped according to the number of '1's in the vector patterns):

- Group 2 (with two '1's) : $[1,1,0,0,0]$
- Group 3 (with three '1's) : $[1,1,0,1,0]$, $[1,0,1,1,0]$
- Group 4 (with four '1's) : $[0,1,1,1,1]$

The vector pattern in Group 2 has the lowest number of '1's, hence it is passed as a matching rule. In Group 3, the first vector pattern matches the first and second attributes

in vector pattern $[1,1,0,0,0]$, so it is filtered out. Meanwhile, the second vector pattern is passed as the second matching rule and we use it to filter out more vector patterns in the next groups. For the vector pattern in Group 4, since it neither matches $[1,1,0,0,0]$ nor $[1,0,1,1,0]$, we declare it as the third matching rule.

Algorithm 12 shows the two filter components in the Cocktail Matcher: (1) Line 1 to 10, which filter out vector patterns that do not satisfy the two decision rules (2) Line 11 to 20, which further filter out similar vector patterns from (1).

4.5 Classifier

The Classifier follows similar approach used in [29], which retains all matching record pairs in the memory to be re-evaluated during each pass in the multi-pass. We utilised a Java Collection object to store the records and the set of records they belong to respectively. All records belonging to the same class (i.e. they are duplicates of each other) are stored in the same set. The obvious disadvantage of this method is the huge consumption of memory to store the data. This only gets worse as the dataset gets larger. An alternative is to store the information in database but that also has the weakness of having to access the database each time a record pair needs to check if they belong to any existing classes.

We illustrate how the classifier works in the following scenario (also refer to Figure 4.2) for the SN method. Suppose the pair (4, 3) is caught by the Matcher. Next, the pair (5, 2) is caught. At this point, the two pairs are stored in two different groups. In the next matching process, the pair (5, 4) is caught. Since record 4 points to Set A whilst record 5 points to the Set B, we join both Sets together. What we get in the end is Set A consisting of records $\{2, 3, 4, 5\}$. Algorithm 13 models this concept.

4.6 Evaluator

The values of true positives, tp, and false positives, fp, can be easily identified by comparing each set of records in the Classifier Collection object with the database. We simply add a count of one to tp if the pair exists. Otherwise, we add a count of one to tn. We used the following SQL statement to load the entire dataset into memory, ordered ascendingly according to the identifiers.

```
SELECT id, class FROM tableName ORDER BY id ASC
```

Input: A collection of vector patterns γ^j and their respective frequency counts $f(\gamma^j)$ (*mCollectionVecPattern*) from Algorithm 5, a collection of vector patterns γ^j and their respective frequency counts $f(\gamma^j)$ (*uCollectionVecPattern*) from Algorithm 8, total attributes of dataset (n), total record pairs selected to compute each of m_i and u_i (N), a list of insignificant attributes computed prior to this (*insignificantList*)

Output: A ruleset

for $\gamma^j \in$ *mCollectionVecPattern for* $j \leftarrow 1$ **to** *total collection size* **do**
  mFrequency $\leftarrow f(\gamma^j)$ of γ^j in *mCollectionVecPattern*;
  if *uCollectionVecPattern contains* γ^j **then**
    uFrequency $\leftarrow f(\gamma^j)$ of γ^j in *uCollectionVecPattern*;
  else *uFrequency* $\leftarrow 0$;
  if *uFrequency is 0* **then** *minU* $\leftarrow 0$;
  else *minU* $\leftarrow$ *uFrequency*$/N$;
  if $\frac{\log(mFrequency - uFrequency)}{\log N} \geq t1$ *AND minY* $\leq t2$ **then**
    modify γ^j to change from 1 to 0 for insignificant attributes based on *insignificantList*;
    count the total number of attributes with 1 and store γ^j together with the total count in list X;
for $x \leftarrow 1$ **to** n **do**
  $A \leftarrow$ all the γ^j in X with x total number of attributes with 1;
  store A into list Z;
  for $y \leftarrow x+1$ **to** n **do**
    $B \leftarrow$ all the γ^j in X with y total number of attributes with 1;
    for $a \in A$ *and* $b \in B$ **do**
      if *all the 1s in a are also in b and at the same position* **then** continue;
      else store b into list Y;
  clear off X;
  copy Y into X;
return a ruleset Z consisting of γ^j as matching rules;

Algorithm 12: cocktailMatcher()

Input: Record pair $(t1, t2)$

Output: A collection of records and the set that they belong to respectively (c)

```
1  c ← a collection of records and the set that they belong to respectively
2  if both t1 and t2 exist in c then
3      if both t1 and t2 do not belong to the same set then
4          add the set of t2 to the set of t1;
5          change all the records in the set of t2 to point to the set of t1;
6          empty the set of t2 from c;
7  else if t1 exists in c but not t2 then
8      add t2 to the set of t1;
9      let t2 points to the set of t1 in c;
10 else if t2 exists in c but not t1 then
11     add t1 to the set of t2;
12     let t1 points to the set of t2 in c;
13 else
14     create a new set in c;
15     add t1 and t2 to the new set;
16     let t1 and t2 point to the new set;
```

Algorithm 13: classifier()

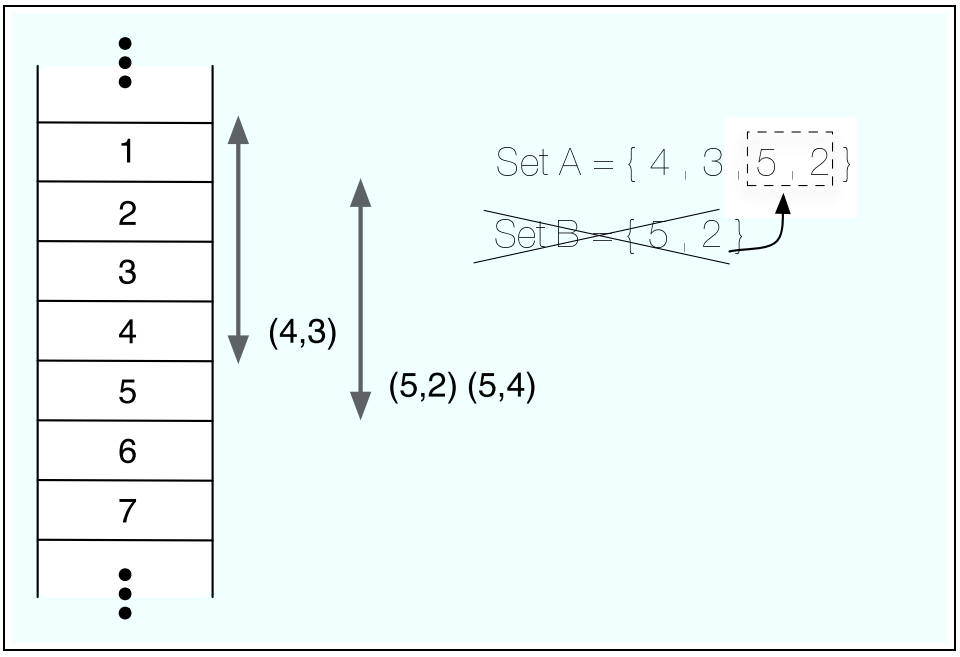

Figure 4.2: How the Classifier Works

Algorithm 14 shows how we evaluate the sets of matching record pairs from the Classifier. Line 3 to 11 shows that we evaluate sets of matching record pairs one at a time. Then, we iterate through the set to retrieve from data in the memory, the class which each record belongs to. We may find that the records in a set may belong to more than one classes. For example, suppose that we have a set of records $\{1, 2, 4, 5\}$ found to be duplicates of one another by the Matcher. Suppose that after retrieving information for the four records from memory, we found that records 1 and 2 belong to class A whilst records 4 and 5 belong to class B. Consequently, the number of duplicates from the set $\{1, 2, 4, 5\}$ is two.

Having found true positives, false positives, and false negatives (total duplicates - true positives), we calculate the Precision, Recall and F-Score values of the matching process using Equation (3.1), (3.2), and (3.3) respectively.

Input: A collection of records and the set that they belong to respectively (c), data from SQL ($data$)

Output: true positives (tp), false positives (tp)

```
1  create a collection classes to store classIDs and their respective group of
   records;
2  for s ∈ c do
3      for r ∈ s do
4          rewind to the beginning of data;
5          move forward to record r in data;
6          get the class of r;
7          if class exists in classes then
8              add r to the group of records that the class belongs to;
9          else
10             add a new entry of class with r to classes;
11         break from the inner loop;
12     for group of records g ∈ classes do
13         if size of group ≥ 1 then
14             add (size of group - 1) to tp;
15         else
16             fp ← fp + 1;
17     if tp has not been increased and fp has been increased then
18         fp ← fp − 1;
```

Algorithm 14: evaluator()

Chapter 5

Experimental Study

The following experiments were performed on a personal MacBook 2GHz Intel Core Duo with 2GB 667 MHz memory. Memory activity during experiments was averagely at 500M inactive and 500M free.

5.1 Experiments Set-Up

There are a few datasets used by previous researches in record matching that are available for download on the Internet. We have selected three different datasets in our experiments, all of which were pre-labelled and downloadable from the Repository of Information on Duplicate Detection, Record Linkage, and Identity Uncertainty (RIDDLE) [1] website. The first two datasets, Restaurant and Cora, were chosen because they are real-world data. The third dataset, DBGen, consists of synthetic data and is useful to run experiments that analyse the efficiency of the system we have implemented.

The Restaurant dataset, used in [30, 31, 3], is an integration of the restaurants reviewed at the Zagat's website [2] with the restaurants reviewed at the Fodor's website [3]. The dataset has the following atrributes: name, address, city, phone, and type of food. There are a total of 864 restaurant records altogether with 112 duplicates.

The second dataset of interest is the "Cora" dataset, which was also used in [3, 32]. It is a collection of 1295 citations to 122 computer science research papers from the Cora Computer Science Research Paper Engine. The citations have the attributes of author,

[1] http://www.cs.utexas.edu/users/ml/riddle/index.html
[2] http://www.zagat.com/
[3] http://www.fodors.com/

volume, title, institution, venue, address, publisher, year, pages, editor, note, and month.

The third dataset is DBGen (UIS Database Generator) [7, 33], a handy tool that can be used to generate huge sets of randomly pertubed names and US mailing addresses. Datasets of varying sizes and data contamination can be generated using command-line or the terminal user interface that provides options such as the number of records to generate, the type of distributions to corrupt the data, and etc. Each record contains ten fields: social security number (SSN), first name, middle name, last name, street number, street address, apartment number, city, state, and zip code. All field values are chosen randomly and independently from a pool of real-world data each time the generator is run.

Both the Restaurant and Cora dataset are stored in ARFF format. The problem with the datasets is that records belonging to the same classes (i.e. they are duplicates of each other) are grouped together. We solved this problem by re-sorting the data. This is achieved by using shell scripts to append a random number to each record, sort the records based on the random numbers, then remove the numbers. We also added attributes 'candidate key' and 'record identifier' to the ARFF files so that each line of the records has additional two fields: a blank candidate key and an increasing identifier number. Finally, we wrote a Java program to extract out data from the files using the Weka Java library 3.2.4, and stored them in the MySQL database.

For DBGen, we used the shell scripts provided by Hernandez [7] to generate datasets of varying sizes and clusters. For example, the following command line creates a dataset of around 50,000 records and exactly 10,000 clusters.

```
sh dbgen -q -n 50000 -c 10000
```

DBGen also include a shell script that randomises the dataset such that records in similar clusters are not grouped together in the output data file. Similar to the first two datasets, we created a 'candidate key' field and a 'record identifier' for each record, then transferred the data into the MySQL database.

5.1.1 Thresholds Selections

There are a few threshold values for experiments on the three approaches. These values were set based on empirical results.

For the Rule-based approach, the threshold we need to set is the similarity threshold, used to compare a record pair with the Equational Theory rules (refer to Section 4.4.2).

The reasonable value for this threshold is 0.8.

For the Probabilistic-based approach, the similarity threshold is also required to compute the vector pattern of a record pair. We also set this threshold to 0.8. Therefore, the binary value of the i-th attribute of a record pair is set to one if the attribute has similarity score greater than 0.8, otherwise, it is set to zero. The Probabilistic-based approach also has a blocking key similarity threshold to compare between two candidate keys. This comparison is required to create blocks for the EM algorithm (refer to variable $threshold$ in Algorithm 1). This threshold was set to 1.0. Meanwhile, as suggested in Section 4.4.1.1, the values of m_i were estimated from the vector patterns and their frequency counts over all the blocks. To reduce the time complexity of the system, it is possible to consider lesser blocks (or record pairs) for larger datasets such as Cora and DBGen. We decided to set a limit of 1,000 record pairs to estimate both m_i and u_i.

The Cocktail approach has three threshold values in addition to the threshold values of the previous two approaches. The first additional threshold is the relative weight threshold to determine if an attribute is significant or not. This threshold is set to 0.6. Therefore, an attribute is considered insignificant if its weight compared to the attribute with the highest weight is less than 0.6. The two other threshold values for the Cocktail approach are the thresholds for Decision Rule 1 in Equation (4.4) and Decision Rule 2 in Equation (4.5). For the experiments, we set the former to 0.5 and the latter to 0.02.

5.2 Comparisons Between the Three Approaches

In the following experiments, we compare between the Rule-based approach, the Probabilistic-based approach, and the Cocktail approach. We show that statistics from the Probabilistic-based approach can assist in designing a good ruleset for the Rule-based approach.

The Cocktail approach consists of two parts:

1. Leveraging the EM algorithm to find insignificant attributes that can be ignored when manually designing rules.

2. Leveraging the EM algorithm again to automatically create rules based on the information obtained in 1.

Experiments on Part 1 and Part 2 in this chapter shall be referred to as Rule+EM(1) and Rule+EM(2) respectively.

Similar experiments were carried out for the Restaurant, Cora, and DBGen datasets. In each of the following sections, we mainly compare between the Rule-based approach, Rule+EM(1), and Rule+EM(2). This is to show the improvements of Rule+EM(1) and Rule+EM(2) over the Rule-based approach after leveraging statistics from the EM algorithm. In the last part of each section, we make an overall comparison between the Rule-based, Rule+EM(1), Rule+EM(2), and the Probabilistic-based approaches.

5.2.1 Experiments on Restaurant Dataset

Table 5.1 shows values computed from the EM algorithm that determine the significance of individual attributes in the Restaurant dataset. As explained in Section 2.3.3, if the weight score of an attribute is low in contrast to the attribute with the highest weight, we can assume that the attribute is not important. To avoid biasing on any particular attributes, we considered each attribute as the key for the sorting and blocking phases. It is obvious from Table 5.1 that attributes 'city' and 'type' display mostly relatively low weights compared to the other fields for varying attributes as the blocking key. The low weight of 'city' is due to the high value of u_i, which means that most record pairs agree on 'city' even when they are non-matching. Meanwhile, most record pairs do not agree on 'type' given that they are matching pairs. Hence, this resulted in the low value of m_i for 'type'. Subsequently, we make a conclusion that attributes 'city' and 'type' are trivial.

After analysing the data, we noticed that the types of restaurant for matching record pairs differ most of the time. For example, the Zagat website has restaurant groups 'Thai food', 'Chinese food' and 'Japanese food' whereas the Fodor's Travel Guides puts them all under the category of Asian restaurants. We can derive inferencing rules, similar to the ILFD (Instance Level Functional Dependency) proposed by Lim et. al. [8] to further correlate a record pair. The following rule is an example of relating Restaurant A from Zagat and Restaurant B from Fodor. Note that this rule is only useful for this instance of data and may not be applicable for datasets from other restaurant guides.

```
similar_name(A.name, B.name) AND similar_phone(A.phone, B.phone)
AND A.type=("Chinese" OR "Thai" OR "Japanese" OR "India")
-> A.type="Asian"
```

Meanwhile, there is also a mixture between the type of food (i.e. steak, seafood, bbq), where the food originates from (i.e. america, california, south-western, greece, mediter-

Table 5.1: Computed values from the EM algorithm that show the significance of each attribute *i* in the Restaurant dataset for varying blocking keys.

Blocking Key	*i*	m_i	u_i	**Weight when *i* agrees**	**Relative Weight**
name	name	0.99999	0.012195121951219513	4.406709247214253	1.0
	address	0.7297725842767431	0.024390243902439025	3.3985497449415356	0.7712216881769461
	city	0.8261576425774451	0.2073170731707317	1.382536230115244	0.3137343883055658
	phone	0.9855678448245216	0.012195121951219513	4.392181935554046	0.9967033650633087
	type	0.488945214628286	0.13414634146341464	1.2933191431571918	0.2934886489220447
	Insignificant attributes: city, phone				
address	name	0.8841795091855169	0.010416666666666666	4.441253019197998	0.9730311608415841
	address	0.99999	0.010416666666666666	4.564338191417836	0.9999978091027277
	city	0.8068788974053298	0.14583333333333334	1.7107091747064114	0.3747981317280403
	phone	0.9999999999680015	0.010416666666666666	4.564348191435838	1.0
	type	0.42147924459738245	0.11458333333333333	1.3024681741549788	0.2853568832892332
	Insignificant attributes: city, phone				
city	name	0.8129191153814787	0.010801080108010801	4.320985476392265	0.7305870124599599
	address	0.5798817394231093	0.0054005400540054005	4.676325226633609	0.790667428826887
	city	0.99999	0.1701170117011701	1.7712587745299875	0.29948229713972707
	phone	0.9999987573489998	0.002700270027003	5.914402258319748	1.0
	type	0.11613462123966528	0.10081008100810081	0.14151168550493243	0.023926625096537684
	Insignificant attributes: city, phone				
phone	name	0.7817984008634408	0.009708737864077669	4.388570617149465	0.9468903353886297
	address	0.99999	0.009708737864077669	4.634718988179635	1.0
	city	0.8828319460682346	0.1650485436893204	1.6768952261282692	0.3618116287967004
	phone	0.99999	0.009708737864077669	4.634718988179635	1.0
	type	0.3289302727016085	0.11650485436893204	1.0379128506858566	0.2239429948898616
	Insignificant attributes: city, phone				
type	name	1.997782621316375E-6	0.01572052401746725	-8.970684523704637	-2.8701664127779956
	address	0.09943771183851076	0.005240174672489083	2.943176604242314	0.9416680091745938
	city	0.9909001650998641	0.16069868995633188	1.8190826670781275	0.5820146678124519
	phone	0.039774872411680366	0.0017467248908296944	3.12549282291337	1.0
	type	0.99999	0.09344978165938865	2.3703210814764333	0.7583831465231082
	Insignificant attributes: name, city				

ranean, caribbean, pacific, international), the type of restaurants (i.e. delis, cafeterias), and style of food (i.e. eclectic) which makes it inherently difficult to relate two restaurants based on type. The inferencing rules are not sufficient to correlate ”Seafood” with ”Asian”, ”Pacific-Rim” or ”French”, ”International” with ”Californian” or ”American”, or ”Steakhouse” with ”American”. The ILFDs also require analysis of the data, which may not always be available due to privacy issues.

Our first contribution suggests to re-design the rules in Algorithm 15 in the Appendix, which was crafted without any prior knowledge of the data behaviour, to ignore ’city’ and ’type’ completely. By ignoring attribute ’city’, we ended up with just $similar\_address$ in the first rule of Rule+EM(1) (refer to Algorithm 16 in Appendix). By ignoring ’type’, the second and third rules of the Rule-based approach are conflated to the second rule of Rule+EM(1). Both the first and second rules of Rule+EM(1), when assessed separately, display better results than the original rules, as clearly shown in Table 5.4.

Our second contribution involves creating a new set of rules via observation from the EM algorithm. As discussed in Section 2.3.3, the vector patterns in the EM algorithm were computed from matching record pairs belonging to the same block. Since we are concerned with using vector patterns to harness good matching rules, it is highly recommended to choose appropriate blocking attributes for the sorting and blocking phases. This can be achieved by making use of the knowledge from our previous experiments, which has revealed that ’city’ and ’type’ attributes are probably not the most useful blocking attributes.

In the next experiments, we tried different combinations of blocking keys, as shown in Table 5.2. From the table, SQL statement 1 only uses significant attributes, SQL statement 2 and SQL statement 3 use a combination of significant and insignificant attributes, and finally SQL statement 4 only uses insignificant attributes. The EM results from each blocking key are as displayed in Table 5.3.

The results confirm our earlier claim that it is important to choose significant attributes for the blocking keys. In the Restaurant dataset, these attributes are ’name’, ’address’, and ’phone’ based on the results in Table 5.1. For the first blocking key, vector patterns $[1,1,1,1,1]$ and $[1,1,1,1,0]$ (in the order of ’name’, ’address’, ’city’, ’phone’, ’type’) are found to satisfy the decision rules with 21 matching record pairs and 0 non-matching record pair, and 26 matching record pairs and 0 non-matching record pair respectively out of the 86 selected record pairs. After modifying the vector patterns to ignore insignificant

Table 5.2: SQL statements to create blocking keys for the Restaurant dataset.

No.	**SQL Statement**
1	UPDATE Restaurant set candKey=UPPER(CONCAT(SUBSTRING(name, 1, 4), SUBSTRING(addr, 1, 4)))
2	UPDATE Restaurant set candKey=UPPER(CONCAT(SUBSTRING(addr, 1, 3), SUBSTRING(type, 1, 3), SUBSTRING(phone, 1, 3)))
3	UPDATE Restaurant set candKey=UPPER(CONCAT(SUBSTRING(city, 1, 3), SUBSTRING(phone, 1, 3), SUBSTRING(type, 1, 3)))
4	UPDATE Restaurant set candKey=UPPER(CONCAT(SUBSTRING(type, 1, 4), SUBSTRING(city, 1, 4)))

Table 5.3: Matching rules derived from the EM algorithm for the Restaurant dataset based on the blocking keys in Table 5.2.

Blocking Key	**γ^j that satisfied decision rules**	**Insignificant Attribute(s)**	**γ^j after removing insignificant attribute(s)**	**Filtered γ^j**
1	[1, 1, 1, 1, 1] (21, 0)/86 [1, 1, 1, 1, 0] (26, 0)/86	city, type	[1, 1, 0, 1, 0] [1, 1, 0, 1, 0]	[1, 1, 0, 1, 0]
	Matching rule: similar_name && similar_address && similar_phone			
2	[1, 0, 1, 1, 1] (9, 0)/69 [1, 1, 1, 1, 1] (22, 1)/69	city, type	[1, 0, 0, 1, 0] [1, 1, 0, 1, 0]	[1, 0, 0, 1, 0]
	Matching rule: similar_name && similar_phone			
3	-	name, address city, type	-	-
	Matching rule: -			
4	-	name, address phone	-	-
	Matching rule: -			

Table 5.4: True positives (tp) and false positives (fp) captured by each rule in Rule-based, Rule+EM(1), and Rule+EM(2) for Restaurant dataset. Refer to Appendix for the rulesets.

Approach	**No.**	**Rule**	**tp**	**fp**
Rule-based	1	very_similar_phone && (similar_name || very_similar_address)	104	7
	2	similar_name && very_similar_address && similar_phone	56	2
	3	similar_name && similar_phone && similar_type	41	3
Rule+EM(1)	1	very_similar_phone && (similar_name || similar_address)	106	7
	2	similar_name && similar_phone	104	3
Rule+EM(2)	1	similar_name && similar_phone	104	3

fields, we obtained $[1,1,0,1,0]$. Results from the second blocking key show that it is possible to use insignificant and significant attributes together and still achieve a good result. However, using the insignificant attributes more than the significant ones proves to be unrealiable because none of the vector patterns satisfy the decision rules. The fourth blocking key has the same impact as the third blocking key. No satisfying vector patterns can be created from the third and fourth blocking keys because the important attributes are not given the chance to be in the same block. Hence, this has lowered the frequency counts of the vector patterns in good standing as matching rules. Taking the fourth blocking key as an example, it is found that the highest number of record pairs given that they are matching pairs by the EM algorithm are 1164 out of 1558 pairs when the vector pattern is $[0,0,1,0,1]$. This is expected since records are grouped based on 'city' and 'type' attributes. However, the number of non-matching record pairs for this vector pattern also records quite a high number, at 43 pairs. Subsequently, the vector pattern is not accepted because it does not satisfy the second threshold of our decision rules in Section 5.1.1. Meanwhile, the appropriate vector pattern, $[1,1,0,1,0]$, only reported a frequency count of 1 given that it is a matching pair. Therefore, we make a conclusion that significant attributes are important as elements in the blocking keys to derive good matching rules.

From Table 5.3, we chose the second matching rule as our only matching rule for the Restaurant dataset since record pairs that satisfy the first rule also satisfy the second. Table 5.4 shows the results from Rule+EM(2) with only one matching rule.

The results in Table 5.4 were obtained also by prioritising significant attributes when

Table 5.5: True positives (tp) and false positives (fp) captured by candidate keys created from significant and insignificant attributes for Rule+EM(2) multi-pass SN method for Restaurant dataset.

No.	**SQL to generate candidate key**	**tp**	**fp**
1	UPDATE Restaurant SET candKey=UPPER(CONCAT(SUBSTRING(addr, 1, 3), SUBSTRING(name, 1, 3), SUBSTRING(city, 1, 3)))	93	2
2	UPDATE Restaurant SET candKey=UPPER(CONCAT(SUBSTRING(phone, 1, 3), SUBSTRING(addr, 1, 3), SUBSTRING(type, 1, 3)))	97	3
3	UPDATE Restaurant SET candKey=UPPER(CONCAT(SUBSTRING(name, 1, 3), SUBSTRING(phone, 1, 3)))	97	2
4	UPDATE Restaurant SET candKey=UPPER(CONCAT(SUBSTRING(type, 1, 3), SUBSTRING(name, 1, 3), SUBSTRING(city, 1, 3)))	57	1
5	UPDATE Restaurant SET candKey=UPPER(CONCAT(SUBSTRING(city, 1, 3), SUBSTRING(addr, 1, 3), SUBSTRING(type, 1, 3)))	90	3

designing the candidate keys for multi-pass SN method. These candidate keys were created arbitrarily from a selection of significant and insignificant attributes, with significant attributes as the first elements. Table 5.5 shows the effectiveness of using significant and insignificant attributes as the first elements in the candidate keys. From the results, the fourth candidate key records the lowest true positives detected. It has been discovered in Table 5.1 that matching record pairs often do not agree on 'type' because the information is not correct most of the time (m_i is relatively low). Consequently, matching record pairs will not fall into the same sliding window when sorted according to 'type'. This is the reason for the low number of duplicates detected. Meanwhile, using insignificant attribute 'city' as the first element in the candidate key does not result in a poor result like the 'type' attribute. This is due to two reasons; the 'city' attribute has a high value of m_i, which means that most matching record pairs agree on 'city', and the second element, 'address', helps to bring matching record pairs even closer. Therefore, we conclude that insignificant attributes with low m_i should not be part of the candidate keys because they decrease the chance of matching record pairs falling into the same window.

Figures 5.1 and 5.2 show the overall accuracy results between the Rule-based approach, Rule+EM(1), Rule+EM(2), and the Probabilistic-based approach. All four tech-

niques ran through multi-pass SN method using the first three candidate keys in Table 5.5. The Probabilistic-based approach records the lowest recall score. This results in the lowest F-score value for the Probabilistics-based approach as shown in Figure 5.2. This shows that the Probabilistic-based approach does not work well for small datasets with a low number of attributes because the EM algorithm cannot accurately compute the matching and non-matching weight scores of the attributes. However, the values from the EM algorithm serve as good indicators on the relative importance of the attributes overally, which helps in boosting the results of Rule+EM(1) and Rule+EM(2). Both Rule+EM(1) and Rule+EM(2) also show better F-scores than the Rule-based approach, as displayed in Figure 5.2. Meanwhile, Rule+EM(1) has a higher recall value than Rule+EM(2). Nevertheless, this experiment shows that it is possible to obtain a comparable accuracy using the EM algorithm to automatically derive an Equational Theory ruleset.

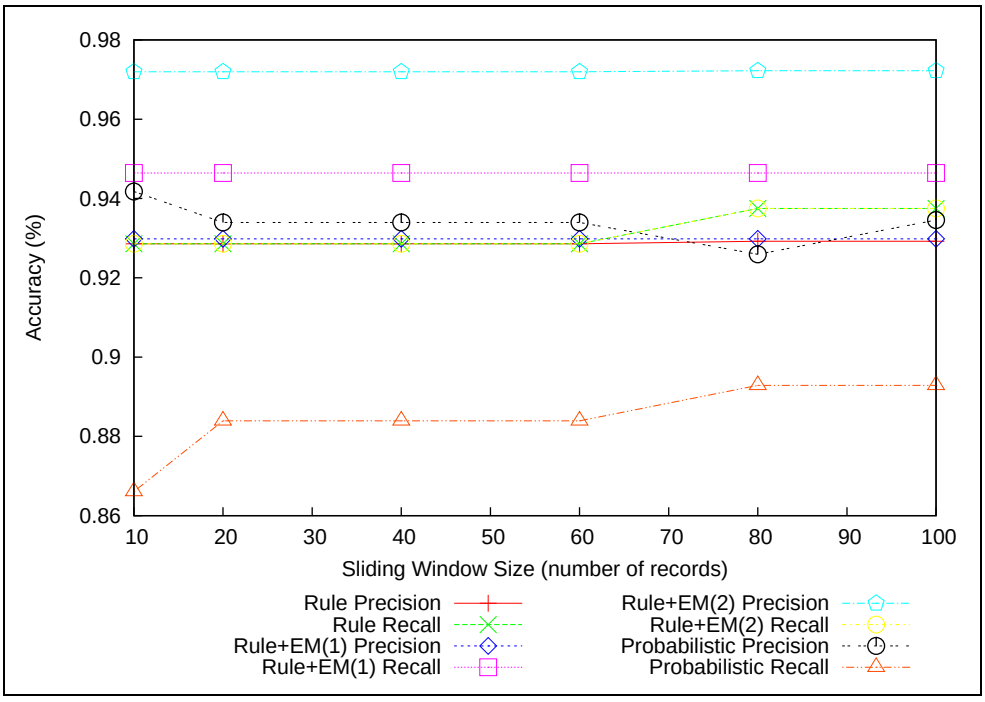

Figure 5.1: Accuracy comparison between Rule-based, Rule+EM(1), Rule+EM(2), and Probabilistic-based approach w.r.t. sliding window size for Restaurant dataset.

5.2.2 Experiments on Cora Dataset

For the Cora dataset, we arbitrarily created a set of SQL statements (refer to Table 5.6) from a combination of significant and insignificant attributes for the blocking keys. The

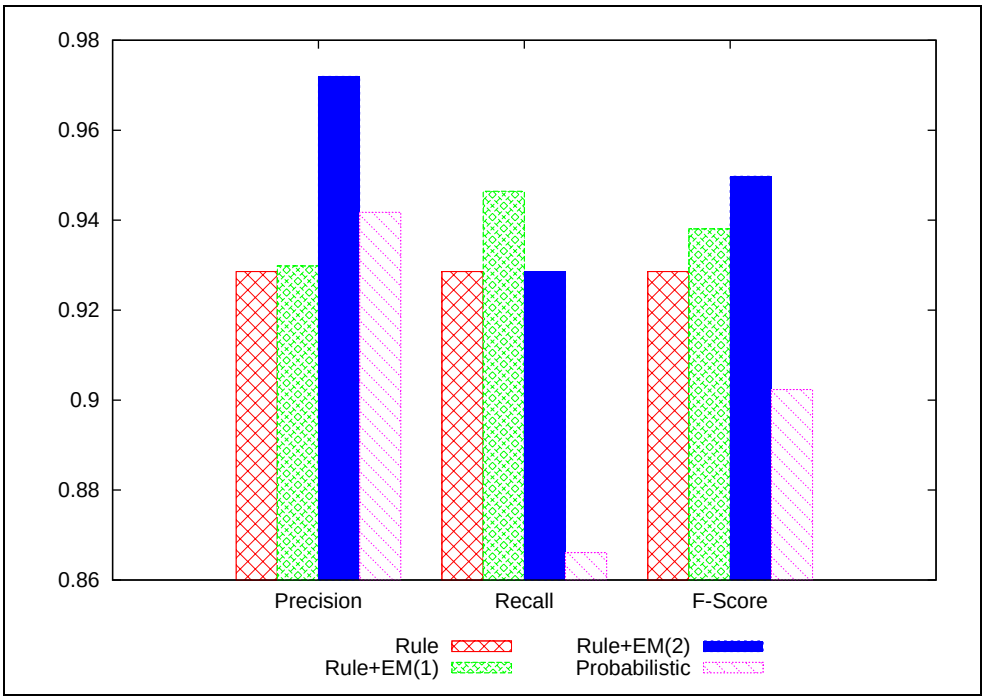

Figure 5.2: Accuracy comparison between Rule-based, Rule+EM(1), Rule+EM(2), and Probabilistic-based approach for Restaurant dataset when sliding window size is 10.

results from the EM algorithm are displayed in Table 5.7. It is obvious that attributes 'institution', 'year', 'editor' and 'note' are trivial because they appear to be insignificant in all scenarios. Attributes 'institution', 'editor', and 'note' display relatively low values of m_i and u_i in almost all cases of blocking keys. Upon analysing the dataset, it is revealed that most data are missing for these attributes. Due to this, not many record pairs that can be found with values that agree to these three attributes. Meanwhile, attribute 'year' exhibits high values of m_i and u_i, which means that the values of 'year' always agree for both matching and non-matching pairs. Since 'year' does not give much impact on the decision-making, we decided to ignore this attribute as well. Note from Table 5.7 that 'editor' appears significant for the fifth blocking key, but that is only because the dataset is sorted according to 'editor' as the first element in the blocking key. Therefore, we can ignore it.

Table 5.6: SQL statements to create blocking keys for the Cora dataset.

No.	**SQL Statement**
1	UPDATE Cora SET candKey=UPPER(CONCAT(SUBSTRING(title, 1, 3), SUBSTRING(author, 1, 3), SUBSTRING(venue, 1, 3))) WHERE title!=”
2	UPDATE Cora SET candKey=UPPER(CONCAT(SUBSTRING(pages, 1, 3), SUBSTRING(publisher, 1, 3), SUBSTRING(venue, 1, 3))) WHERE pages!=”
3	UPDATE Cora SET candKey=UPPER(CONCAT(SUBSTRING(volume, 1, 3), SUBSTRING(title, 1, 3), year)) WHERE volume!=”
4	UPDATE Cora SET candKey=UPPER(CONCAT(SUBSTRING(editor, 1, 3), year, SUBSTRING(note, 1, 3))) WHERE editor!=”
5	UPDATE Cora SET candKey=UPPER(CONCAT(year, SUBSTRING(venue, 1, 3), month)) WHERE year!=”
6	UPDATE Cora SET candKey=UPPER(CONCAT(SUBSTRING(publisher, 1, 3), SUBSTRING(address, 1, 3), SUBSTRING(author, 1, 3))) WHERE publisher!=”
7	UPDATE cora SET candKey=UPPER(CONCAT(SUBSTRING(address, 1, 3), SUBSTRING(author, 1, 3), SUBSTRING(venue, 1, 3))) WHERE address!=”

Table 5.7: Computed values from the EM algorithm that show the significance of each attribute *i* in the Cora dataset based on the blocking keys in Table 5.6.

Blocking Key	*i*	m_i	u_i	**Weight when *i* agrees**	**Relative Weight**
1	author	0.9890702264911804	0.033	3.400257775209813	0.7570504114095382
	volume	0.4462560124476959	0.0050	4.491454893841133	1.0
	title	0.9975892054721215	0.025	3.6864657489420125	0.8207731873244559
	institution	1.0E-5	0.0010	-4.605170185988091	-1.0253181418570827
	venue	0.9000370181763511	0.044	3.0182462598672375	0.671997455436095
	address	7.48186314267994E-137	0.0050	-308.1433585294199	-68.60657978597509
	publisher	0.024833634822820444	0.023	0.0767047612308185	0.01707793199392901
	year	0.9367898816561265	0.54	0.5508898717020045	0.12265287857113011
	pages	0.9599766521789918	0.015	4.158858762417037	0.9259491324559074
	editor	1.0E-5	0.0010	-4.605170185988091	-1.0253181418570827
	note	0.015607725581911482	0.0020	2.0546188407195434	0.45745062330179936
	month	0.042839391410865214	0.0040	2.3711636782631667	0.5279277504299561
	Insignificant attributes: institution, address, publisher, year, editor, note, month				
2	author	0.624003369081501	0.021	3.3916333298005013	0.6162942599249531
	volume	0.9819729294854033	0.0040	5.503269380139131	1.0
	title	0.9469208652252402	0.027	3.5573786600420036	0.6464118716194968
	institution	1.0E-5	0.0010	-4.605170185988091	-0.8368062451399871
	venue	0.8344016831948158	0.055	2.7193817357037537	0.4941393102648745
	address	1.0E-5	0.0020	-5.298317366548036	-0.9627581353130286
	publisher	2.0375190480303323E-118	0.021	-267.13007521722295	-48.540250670133375
	year	0.99999	0.552	0.5941972326550413	0.10797167858063657
	pages	0.99999	0.019	3.9633062997656965	0.7201730509629347
	editor	1.0E-5	0.0010	-4.605170185988091	-0.8368062451399871
	note	1.0E-5	0.0010	-4.605170185988091	-0.8368062451399871
	month	0.2877282521710964	0.0030	4.563404177176164	0.8292169366895127
	Insignificant attributes: institution, venue, address, publisher, year, editor, note				
3	author	0.7245045495356363	0.023	3.449993825364164	0.5031519932239974
	volume	0.9502857335387875	0.0010	6.8567627115178045	1.0
	title	0.99999	0.022	3.816702825573821	0.5566333539824306
	institution	1.0E-5	0.0010	-4.605170185988091	-0.6716245522471488
	venue	0.8994086352240347	0.034	3.2753769507684565	0.47768562054314156
	address	0.0011532308176362288	0.0040	-1.2437269511115776	-0.18138690274674354
	publisher	0.00634434981853942	0.033	-1.6489429370180773	-0.24048417692043325
	year	0.9988467691823638	0.533	0.6280799585163113	0.09160007206626526
	pages	0.99999	0.015	4.199695077829927	0.6124894873167176
	editor	0.0023064616352724576	0.0010	0.8357145905682583	0.12188180132943022
	note	0.005766154088181144	0.0010	1.7520053224424132	0.2555149414022221
	month	0.153495829412789	0.0040	3.647379035588994	0.5319389322693383
	Insignificant attributes: author, title, institution, venue, address, publisher, year, editor, note, month				
4	author	0.6437070653451845	0.043478260869565216	2.694982692091352	0.7039023643273369
	volume	0.26460588444938904	0.021739130434782608	2.499127607978281	0.6527469869005023
	title	0.9999412397047172	0.06521739130434782	2.7299703457992486	0.7130407874569277
	institution	1.0E-5	0.021739130434782608	-7.6842840684811335	-2.007057685320158
	venue	0.99999	0.10869565217391304	2.2191934840049945	0.5796310102009312

Continued on next page

Table 5.7 – continued from previous page

Blocking Key	i	m_i	u_i	Weight when i agrees	Relative Weight
	address	0.09198830315079222	0.021739130434782608	1.44254754680776	0.37677890542020787
	publisher	0.99999	0.021739130434782608	3.8286313964390946	1.0
	year	0.99999	0.6521739130434783	0.42743401477693926	0.11164146414681861
	pages	0.73549098607699	0.043478260869565216	2.8282772214096528	0.7387175542780525
	editor	0.6436915038778712	0.021739130434782608	3.3881056975936485	0.8849391196929621
	note	1.0E-5	0.021739130434782608	-7.6842840684811335	-2.007057685320158
	month	1.0E-5	0.021739130434782608	-7.6842840684811335	-2.007057685320158
	Insignificant attributes: institution, venue, address, year, note, month				
5	author	0.809893701976212	0.039	3.0333413607970154	0.6713838074609955
	volume	0.14449951582768092	0.0030	3.874663868205275	0.8575977020218021
	title	0.9999999999997512	0.025	3.6888794541136876	0.8164771578879408
	institution	0.010411685550325518	0.0020	1.6497816054176602	0.36515397509807107
	venue	0.0.7161838363928708	0.033	3.077429327276955	0.6811420058560278
	address	0.10303829557412002	0.0070	2.689190571763687	0.5952112836336109
	publisher	0.1025048685837748	0.02	1.6341780222747504	0.36170036015188894
	year	0.99999	0.539	0.6180297080231395	0.13679144189283
	pages	0.8187580425473018	0.012	4.222881960071753	0.9346704612455415
	editor	0.003351025908670973	0.0010	1.2092665403489709	0.2676526897326697
	note	0.012647575606628051	0.0020	1.8443183645961032	0.40821171721583527
	month	0.27496833833499534	0.0030	4.518043669042822	1.0
	Insignificant attributes: institution, address, publisher, year, editor, note				
6	author	0.960584329024467	0.027466937945066123	3.5545587505215925	0.7573607149345 61
	volume	0.023103175546894197	0.006103763987792472	1.3310644495791673	0.28360648786838283
	title	0.9049221712529696	0.024415055951169887	3.612648952203414	0.7697378451958012
	institution	0.00231031755468942	0.001017293997965412	0.8202388258131768	0.17476618256592205
	venue	0.4977176542042832	0.0508646998982706	2.2808637925501145	0.4859777974806652
	address	0.3333217484409602	0.003051881993896236	4.69334978752987	1.0
	publisher	0.8997080235914734	0.017293997965412006	3.9517107895664494	0.8419808811322906
	year	0.7053325331694544	0.5198372329603256	0.30515362082291836	0.06501829921854642
	pages	0.606598419214368	0.011190233977619531	3.9928255576407707	0.8507410992996138
	editor	0.013861905328136519	0.001017293997965412	2.6119982950412317	0.5565317765109379
	note	0.00924127021875768	0.001017293997965412	2.2065331869330675	0.4701403660123051
	month	1.0E-5	0.00508646998982706	-6.231754257257163	-1.3277838940994342
	Insignificant attributes: volume, institution, venue, year, editor, note, month				
7	author	0.9999999999999996	0.02167630057803468	3.831535754515459	0.7014607265292825
	volume	0.0038138039353723054	0.004335260115606936	-0.12815451273777173	-0.02346196495937745
	title	0.9999999999998259	0.011560693641618497	4.46014441393766	0.8165436371927789
	institution	0.015255215741489222	0.001445086705202312	2.3567521370502287	0.43146382344375495
	venue	0.6092811647134843	0.03901734104046243	2.7482736544150312	0.5031418621254773
	address	0.680985287798665	0.002890173410404624	5.4622241981718584	1.0
	publisher	0.6774687552252112	0.031791907514450865	3.059151657218851	0.5600560405855755
	year	0.7197049878485174	0.5404624277456648	0.2864162681658186	0.05243583159066937
	pages	0.5796981981765904	0.008670520231213872	4.202578827548559	0.7693896616244922
	editor	0.007627607870744611	0.001445086705202312	1.6636049564902833	0.3045654839739226
	note	0.011441411806116917	0.001445086705202312	2.0690700645984474	0.3787962539675579
	month	0.16780015316203128	0.001445086705202312	4.7546043834336045	0.870452074271304
	Insignificant attributes: volume, institution, venue, publisher, year, editor, note				

Here, we make a comparative study with the work by Parag Singla and Pedro Domingos [32], who also used the Cora dataset in their record matching experiments. Singla and Domingos chose 'author', 'title' and 'venue' as the most informative attributes to support their work while we claim that 'author', 'title' and 'pages' are the most significant. In contrast to their work, our claim is supported by the results in Table 5.7, which clearly show that 'pages' is a more expressive attribute than 'venue' because the weight of 'pages' is higher than 'venue' in all cases. In addition, we claim that 'volume', 'venue', 'address', 'publisher' and 'month' are also important attributes because these attributes exhibit high values of m_i for certain candidate keys. This tells us that there exists some combinations of attributes for record pairs that agree on either one of the attributes, and the combinations can be converted into matching rules to identify matching record pairs.

Since 'institution', 'year', 'editor', and 'note' are trivial attributes in all situations, for experiments on Rule+EM(1), we modified the original ruleset (refer to Algorithm 18 in the Appendix) to exclude these four attributes. The resulting true positives and false positives for each modified rule are displayed in Table 5.9. The results of Rule+EM(1) show significant improvements over the original ruleset.

For experiments on Rule+EM(2), we used the same blocking keys in Table 5.6 to create a new set of matching rules. The vector patterns that satisfy the decision rules are displayed in Table 5.8. We further modified these vector patterns to ignore insignificant attributes for the respective blocking keys. For instance, attributes 'author', 'title', 'institution', 'venue', 'address', 'publisher', 'year', 'editor', 'note', and 'month' are found to be insignificant for the third blockingkey (refer to Table 5.7). Hence, we consider these attributes not important in determining a potential mapping between two records that agree on attributes 'volume' and 'pages'.

Meanwhile, a few vector patterns consist of only zeroes after considering the insignificant attributes. In these cases, we ignore contribution from these particular vector patterns. From Table 5.8, we were able to obtain six vector patterns, hence six matching rules. The results for these automatically derived rules are favourable (refer to Table 5.9).

Table 5.8: Matching rules derived from the EM algorithm for the Cora dataset based on the blocking keys in Table 5.6.

Blocking Key	**γ^j that satisfied decision rules**	**γ^j after removing insignificant attribute(s)**	**Filtered γ^j**
1	[1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0] (84,1)	[1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]	[1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
	[1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0] (37,0)	[1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0]	
	[1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0] (98,0)	[1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]	
	[1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0] (91,1)	[1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0]	
	[1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0] (46,0)	[1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0]	
	[1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0] (167,0)	[1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0]	
	[1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0] (70,3)	[1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]	
	[1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0] (141,1)	[1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0]	
	Matching rule: similar_author && similar_title		
2	[1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1] (51,0)	[1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1]	[0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0]
	[1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0] (69,1)	[1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0]	
	[1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0] (138,3)	[1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0]	
	[0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0] (103,5)	[0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0]	
	[0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0] (98,1)	[0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0]	
	[1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0] (187,0)	[1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0]	
	[0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0] (173,3)	[0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0]	
	[0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1] (48,0)	[0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0]	
	Matching rule: similar_title && similar_pages		
3	[1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1] (77,0)	[0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]	[0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
	[0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0] (171,0)	[0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]	
	[1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0] (480,0)	[0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]	
	Matching rule: similar_volume && similar_pages		
4	-	-	-
	Matching rule: -		
5	[1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0] (35,1)	[1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0]	[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
	[1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0] (50,0)	[1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]	[1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
	[1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0] (71,1)	[1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0]	
	[1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1] (56,0)	[1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1]	
	[1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0] (106,1)	[1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0]	
	[1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0] (43,4)	[1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]	
	[1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0] (49,0)	[1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0]	
	[0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0] (65,15)	[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]	
	[0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1] (33,0)	[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1]	
	Matching rule: (i) similar_venue (ii) similar_author && similar_title		
6	[1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0] (46,0)	[1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0]	[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
	[0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0] (57,2)	[0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0]	
	[1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0] (54,0)	[1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0]	
	[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0] (85,8)	[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]	
	[0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0] (127,5)	[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]	
	[0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0] (35,0)	[0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0]	
	[1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0] (45,0)	[1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0]	
	[1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0] (51,0)	[1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0]	
	[0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0] (35,0)	[0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0]	
	Matching rule: similar_publisher		

Continued on next page

Table 5.9: True positives (tp) and false positives (fp) captured by each rule in Rule-based, Rule+EM(1), and Rule+EM(2) for Cora dataset. Refer to Appendix for the rulesets.

Approach	No.	Rule	tp	fp
Rule-based	1	similar_author && similar_title && similar_year	893	9
	2	similar_title && similar_venue && similar_year	726	3
	3	similar_author && similar_venue && similar_date	93	0
	4	(similar_author || similar_title) && similar_pages && similar_volume && similar_publisher	10	1
	5	similar_title && similar_date	108	1
Rule+EM(1)	1	similar_author && similar_title	1066	12
	2	similar_title && (similar_venue || similar_month)	849	5
	3	similar_author && similar_venue && similar_month	93	0
	4	(similar_author || similar_title) && similar_pages && similar_volume && similar_publisher	10	1
Rule+EM(2)	1	similar_author && similar_title	1066	12
	2	(similar_title || similar_volume) && similar_pages	734	5
	3	similar_publisher	282	23
	4	similar_venue	881	33
	5	similar_address	143	16

Table 5.8 – continued from previous page

Blocking Key	**γ^j that satisfied decision rules**	**γ^j after removing insignificant attribute(s)**	**Filtered γ^j**
7	[0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0] (53,1)	[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]	[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
	[1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0] (33,0)	[1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0]	
	[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0] (40,7)	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	
	[0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0] (50,11)	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	
	[0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0] (29,0)	[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]	
	[1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0] (50,1)	[1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0]	
	[0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0] (37,0)	[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]	
	Matching rule: similar_address		

The results in Table 5.9 were obtained using the first three candidate keys generated in Table 5.10 for the multi-pass SN method. These three candidate keys are a combination of the significant attributes for the Cora dataset. It is apparent that using the significant

Table 5.10: True positives (tp) and false positives (fp) captured by candidate keys created from significant and insignificant attributes for Rule+EM(2) multi-pass SN method for Cora dataset.

No.	**SQL to generate candidate key**	**tp**	**fp**
1	UPDATE Cora SET candKey=UPPER(CONCAT(SUBSTRING(venue, 1, 3), SUBSTRING(title, 1, 3), SUBSTRING(author, 1, 3)))	992	59
2	UPDATE Cora SET candKey=UPPER(CONCAT(SUBSTRING(title, 1, 3), SUBSTRING(pages, 1, 3), SUBSTRING(volume, 1, 3)))	1052	32
3	UPDATE Cora SET candKey=UPPER(CONCAT(SUBSTRING(author, 1, 3), SUBSTRING(title, 1, 3), SUBSTRING(month, 1, 3)))	994	37
4	UPDATE Cora SET candKey=UPPER(CONCAT(SUBSTRING(institution, 1, 3), SUBSTRING(note, 1, 3), SUBSTRING(month, 1, 3)))	335	301
5	UPDATE Cora SET candKey=UPPER(CONCAT(SUBSTRING(note, 1, 3), SUBSTRING(editor, 1, 3), year))	772	93
6	UPDATE Cora SET candKey=UPPER(CONCAT(SUBSTRING(editor, 1, 3), SUBSTRING(institution, 1, 3), SUBSTRING(author, 1, 3)))	892	60

attributes increases the chance of duplicates falling into the same sliding window. On the other hand, the use of insignificant attributes resulted in poor detection of matching record pairs, as can be seen from candidate keys four to six. The fourth candidate key displays the worst outcome due to the combination of the 'institution', 'note', and 'month' attributes, all of which have very low m_i values. Meanwhile, the scores from the fifth and sixth candidate keys are not as bad due to the use of at least one attribute with high value of m_i in the key combinations.

Figures 5.3 and 5.4 show the accuracy between Rule-based, Rule+EM(1), Rule+EM(2), and Probabilistic-based approach. As expected, Rule+EM(2) records the best accuracy result, followed by Rule+EM(1). The Probabilistic-based approach performs better than the

Rule-based approach because the Equational Theory rules for the Rule-based approach was manually created without analysing the data. An expert would be able to define a ruleset better than the one created in Algorithm 18 in the Appendix, with substantial manual analysis of the Cora dataset. From the results of Rule+EM(2), we show that it is possible to create a good ruleset without the need of an expert.

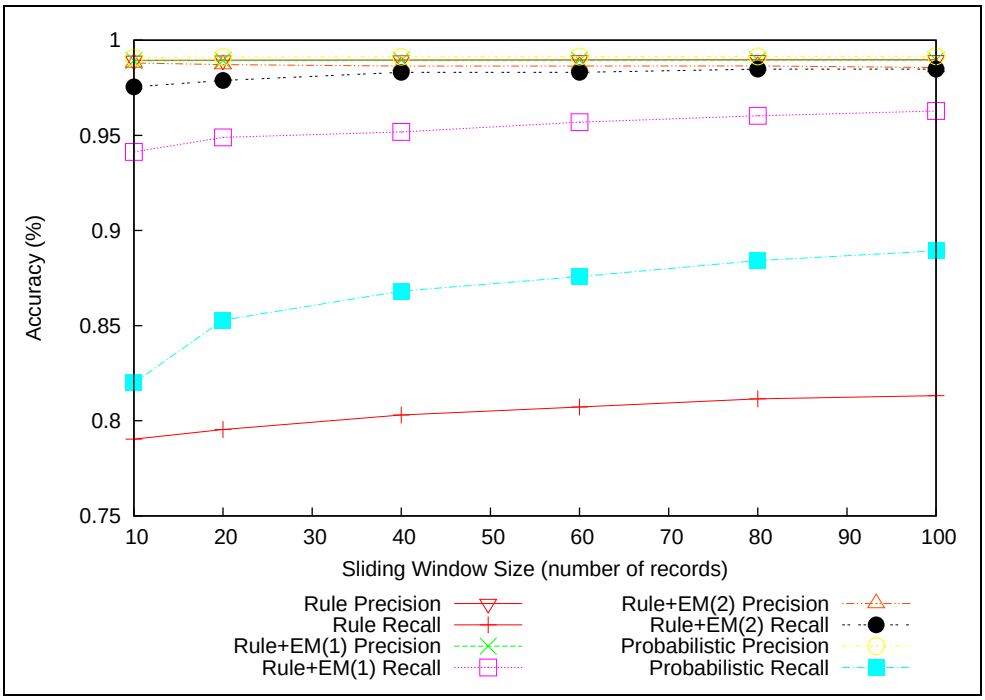

Figure 5.3: Accuracy comparison between Rule-based, Rule+EM(1), Rule+EM(2), and Probabilistic-based approach w.r.t. sliding window size for Cora dataset.

5.2.3 Experiments on DBGen Dataset

Table 5.11 shows the attributes used to create blocking keys to compute values from the EM algorithm. These attributes were arbitrarily chosen. The results are displayed in Table 5.12. It is apparent that 'minit', 'stnum', 'apmt', and 'state' are non-contributional attributes due to their relatively high values of u_i, which means that the probability of non-matching record pairs agree on these four attributes are high. Therefore, we omit these attributes from the original ruleset of DBGen in Algorithm 21 in the Appendix, which is similar to the ruleset designed by Hernandez et. al. [7]. The results we obtained in Table 5.14 clearly shows the better outcome of Rule+EM(1) after re-designing the rules to ignore the trivial attributes.

Table 5.11: SQL statements to create blocking keys for the DBGen dataset.

No.	**SQL Statement**
1	UPDATE DBGen SET candKey=UPPER(CONCAT(SUBSTRING(fname, 1, 3), SUBSTRING(city, 1, 3), SUBSTRING(stadd, 1, 3))) WHERE fname!=”
2	UPDATE DBGen SET candKey=UPPER(CONCAT(SUBSTRING(lname, 1, 3), SUBSTRING(ssn, 1, 3), SUBSTRING(fname, 1, 3))) WHERE lname!=”
3	UPDATE DBGen SET candKey=UPPER(CONCAT(zip, SUBSTRING(city, 1, 3), state)) WHERE zip!=0
4	UPDATE DBGen SET candKey=UPPER(CONCAT(state, apmt, SUBSTRING(stadd, 1, 3))) WHERE state!=”
5	UPDATE DBGen SET candKey=UPPER(CONCAT(SUBSTRING(ssn, 1, 3), stnum, zip)) WHERE ssn!=0

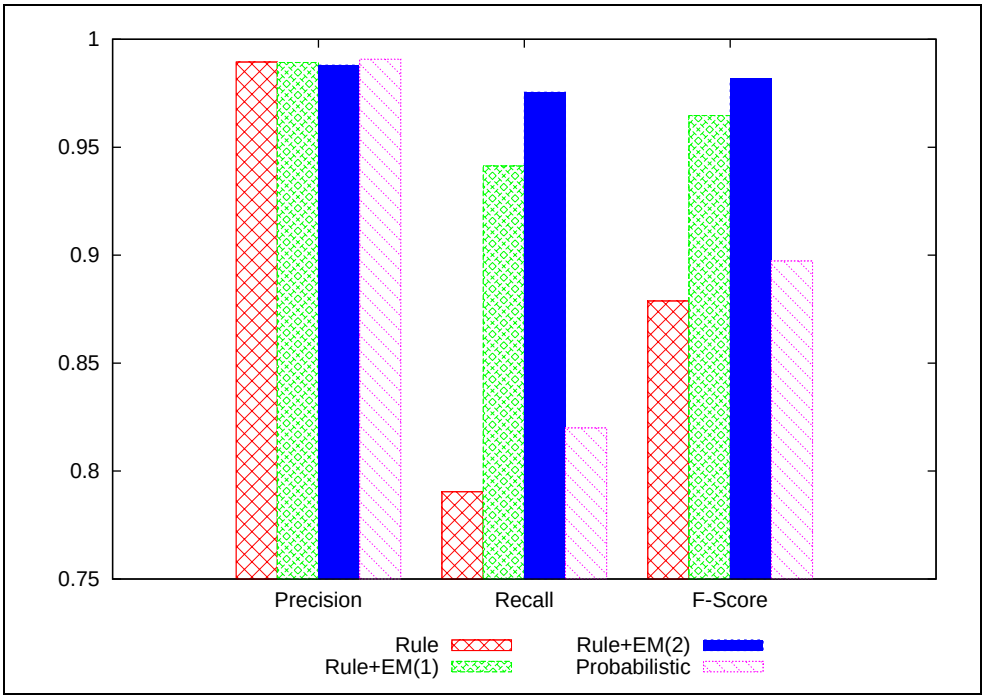

Figure 5.4: Accuracy comparison between Rule-based, Rule+EM(1), Rule+EM(2), and Probabilistic-based approach for Cora dataset when sliding window size is 10.

Table 5.12: Computed values from the EM algorithm that show the significance of each attribute i in the DBGen dataset for blocking keys in Table 5.11.

Blocking Key	i	m_i	u_i	**Weight when i agrees**	**Relative Weight**
1	ssn	0.5925058536882867	0.0010	6.384360752603161	0.9242322197482816
	fname	0.99999	0.0010	6.907745278932136	1.0
	minit	0.7634660451183394	0.024	3.459814820654271	0.5008602200788043
	lname	0.9777517559361134	0.0010	6.885255809528783	0.9967443111326145
	stnum	0.9566744359318059	0.022	3.772420687877337	0.5461146199734385
	stadd	0.9718969565944146	0.0090	4.682025209770437	0.6777935521233099
	apmt	0.999999961182068	0.016	4.135166517924423	0.5986275334350017
	city	0.99999	0.0010	6.907745278932136	1.0
	state	0.9707259955817009	0.019	3.933605261435971	0.5694485107076884
	zip	0.99999	0.01	4.605160185938091	0.666666184114131
	Insignificant attributes: minit, stnum, apmt, state				
2	ssn	0.676567656765744	0.0010	6.517032452611276	0.9434384432916929
	fname	0.99999	0.0010	6.907745278932136	1.0
	minit	0.7469746974697217	0.023	3.480537096434171	0.5038600810961322
	lname	0.99999	0.0030	5.809132990264027	0.8409593515240413
	stnum	0.7403740374038142	0.033	3.1106479529830056	0.45031306560624634
	stadd	0.8789878987899667	0.01	4.476186037576987	0.6479952367712316
	apmt	0.7623762376238384	0.015	3.928389982892451	0.5686935207170434

Continued on next page

Table 5.12 – continued from previous page

Blocking Key	*i*	m_i	u_i	**Weight when *i* agrees**	**Relative Weight**
	city	0.995599559956095	0.0070	4.957434979449371	0.7176632575855109
	state	0.9779977997799758	0.017	4.052294076262345	0.5866305013622023
	zip	0.99999	0.01	4.605160185938091	0.666666184114131
	Insignificant attributes: minit, stnum, apmt, state				
3	ssn	0.7423126851863513	0.0010	6.609770563006653	0.9568636792624254
	fname	0.9503265448923849	0.0010	6.856805656998128	0.9926257237525292
	minit	0.7556395886763306	0.028	3.295360017775163	0.47705291447639925
	lname	0.9260187745397391	0.0010	6.830894509321574	0.9888746955037632
	stnum	0.83844696426968	0.029	3.36425549855447	0.48702657129166005
	stadd	0.8903629677366729	0.0070	4.845719059448873	0.7014906983075626
	apmt	0.9156723425576769	0.021	3.775136158327788	0.54650772515332
	city	0.99999	0.0010	6.907745278932136	1.0
	state	0.99999	0.019	3.9633062997656965	0.5737481826165978
	zip	0.99999	0.011	4.5098500061337665	0.6528686024205775
	Insignificant attributes: minit, stnum, apmt, state				
4	ssn	0.7927736129715873	0.0010	6.67553769912634	0.9678800370490542
	fname	0.943676901232841	0.0010	6.84978384191607	0.9931438241386021
	minit	0.7279489692955202	0.023	3.45473673268908	0.5008990837198704
	lname	0.964930885494842	0.0010	6.872056477525193	0.9963731129764676
	stnum	0.8384697175435354	0.017	3.8983651215203405	0.5652203535217333
	stadd	0.9808713736902954	0.012	4.403534683637685	0.6384644211212886
	apmt	0.8384696791359348	0.029	3.3642825897832984	0.4877842211975081
	city	0.9893729673950972	0.0010	6.89707137619991	1.0
	state	0.99999	0.024	3.729691448584191	0.5407645136824936
	zip	0.99999	0.0070	4.961835129876824	0.719411886470958
	Insignificant attributes: minit, stnum, apmt, state				
5	ssn	0.7085388182496124	0.0010	6.56320484697299	0.9544386748161506
	fname	0.951995616944178	0.0010	6.858560430730534	0.9973900680049698
	minit	0.7661289031886592	0.032	3.175614533708361	0.4618062970680259
	lname	0.9692355306634722	0.0010	6.876507648055263	1.0
	stnum	0.9477756447002329	0.02	3.8583855389836144	0.561096669480881
	stadd	0.9827465907883924	0.0080	4.810909753563349	0.6996152698127104
	apmt	0.99999	0.018	4.017373521035972	0.5842171239600265
	city	0.9897917622667703	0.0030	5.798882291191048	0.8432888594010396
	state	0.99999	0.026	3.6496487409106546	0.5307416100879067
	zip	0.99999	0.013	4.3427959214706	0.6315409134604513
	Insignificant attributes: minit, stnum, apmt, state				

For Rule+EM(2), we refer again to the EM algorithm to estimate the frequency counts of record pairs given that they are matching and non-matching. The vector patterns of record pairs that satisfy the two decision rules in Equation (4.4) and Equation (4.5) for each blocking key in Table 5.11 are presented in Table 5.13. These vector patterns were transformed to ignore insignificant attributes, then filtered to remove redundant vector

patterns. The filtered vector patterns were translated into matching rules. The final ruleset for Rule+EM(2), in Algorithm 23 in the Appendix, was created from these matching rules.

Table 5.13: Matching rules derived from the EM algorithm for the DBGen dataset based on the blocking keys in Table 5.11

Blocking Key	**γ^j that satisfied decision rules**	**γ^j after removing insignificant attribute(s)**	**Filtered γ^j**
1	[1, 1, 1, 1, 0, 1, 0, 1, 1, 1] (70,0) [1, 1, 0, 1, 1, 1, 1, 1, 1, 1] (76,0) [0, 1, 0, 1, 1, 1, 1, 1, 1, 1] (103,0) [1, 1, 0, 1, 0, 1, 0, 1, 1, 1] (48,0) [1, 1, 1, 1, 1, 1, 1, 1, 1, 1] (356,0) [0, 1, 1, 1, 1, 1, 1, 1, 1, 1] (214,0)	[1, 1, 0, 1, 0, 1, 0, 1, 0, 1] [1, 1, 0, 1, 0, 1, 0, 1, 0, 1] [0, 1, 0, 1, 0, 1, 0, 1, 0, 1] [1, 1, 0, 1, 0, 1, 0, 1, 0, 1] [1, 1, 0, 1, 0, 1, 0, 1, 0, 1] [0, 1, 0, 1, 0, 1, 0, 1, 0, 1]	[0, 1, 0, 1, 0, 1, 0, 1, 0, 1]
	Matching rule: similar_fname && similar_lname && similar_stadd && similar_city && similar_zip		
2	[1, 1, 0, 1, 1, 1, 1, 1, 1, 1] (103,0) [0, 1, 0, 1, 1, 1, 1, 1, 1, 1] (45,0) [1, 1, 1, 1, 1, 1, 1, 1, 1, 1] (331,0) [0, 1, 1, 1, 1, 1, 1, 1, 1, 1] (156,10) [1, 1, 1, 1, 0, 0, 0, 1, 1, 1] (57,0) [1, 1, 1, 1, 0, 0, 0, 0, 0, 0] (38,0) [0, 1, 1, 1, 0, 1, 0, 1, 1, 1] (51,0) [1, 1, 0, 1, 0, 1, 0, 1, 1, 1] (59,0)	[1, 1, 0, 1, 0, 1, 0, 1, 0, 1] [0, 1, 0, 1, 0, 1, 0, 1, 0, 1] [1, 1, 0, 1, 0, 1, 0, 1, 0, 1] [0, 1, 0, 1, 0, 1, 0, 1, 0, 1] [1, 1, 0, 1, 0, 0, 0, 1, 0, 1] [1, 1, 0, 1, 0, 0, 0, 0, 0, 0] [0, 1, 0, 1, 0, 1, 0, 1, 0, 1] [1, 1, 0, 1, 0, 1, 0, 1, 0, 1]	[1, 1, 0, 1, 0, 0, 0, 0, 0, 0] [0, 1, 0, 1, 0, 1, 0, 1, 0, 1]
	Matching rules: (i) similar_ssn && similar_fname && similar_lname (ii) similar_fname && similar_lname && similar_stadd && similar_city && similar_zip		
3	[1, 1, 0, 1, 1, 1, 1, 1, 1, 1] (119,0) [0, 1, 0, 1, 1, 1, 1, 1, 1, 1] (49,0) [1, 1, 1, 1, 1, 1, 1, 1, 1, 1] (350,0) [0, 1, 1, 1, 1, 1, 1, 1, 1, 1] (141,0) [1, 1, 1, 1, 0, 0, 0, 1, 1, 1] (35,0) [0, 0, 0, 0, 0, 0, 0, 1, 1, 1] (77,0) [1, 1, 1, 1, 0, 1, 1, 1, 1, 1] (49,0)	[1, 1, 0, 1, 0, 1, 0, 1, 0, 1] [0, 1, 0, 1, 0, 1, 0, 1, 0, 1] [1, 1, 0, 1, 0, 1, 0, 1, 0, 1] [0, 1, 0, 1, 0, 1, 0, 1, 0, 1] [1, 1, 0, 1, 0, 0, 0, 1, 0, 1] [0, 0, 0, 0, 0, 0, 0, 1, 0, 1] [1, 1, 0, 1, 0, 1, 0, 1, 0, 1]	[0, 0, 0, 0, 0, 0, 0, 1, 0, 1]
	Matching rules: similar_city && similar_zip		
4	[1, 1, 0, 1, 1, 1, 1, 1, 1, 1] (141,0) [0, 1, 0, 1, 1, 1, 1, 1, 1, 1] (53,0) [1, 1, 1, 1, 1, 1, 1, 1, 1, 1] (463,0) [0, 1, 1, 1, 1, 1, 1, 1, 1, 1] (186,1)	[1, 1, 0, 1, 0, 1, 0, 1, 0, 1] [0, 1, 0, 1, 0, 1, 0, 1, 0, 1] [1, 1, 0, 1, 0, 1, 0, 1, 0, 1] [0, 1, 0, 1, 0, 1, 0, 1, 0, 1]	[0, 1, 0, 1, 0, 1, 0, 1, 0, 1]
	Matching rules: similar_fname && similar_lname && similar_stadd && similar_city && similar_zip		
5	[1, 1, 1, 1, 0, 1, 0, 1, 1, 1] (107,0) [1, 1, 0, 1, 1, 1, 1, 1, 1, 1] (131,0) [1, 1, 1, 1, 1, 1, 1, 1, 1, 1] (626,0)	[1, 1, 0, 1, 0, 1, 0, 1, 0, 1] [1, 1, 0, 1, 0, 1, 0, 1, 0, 1] [1, 1, 0, 1, 0, 1, 0, 1, 0, 1]	[1, 1, 0, 1, 0, 1, 0, 1, 0, 1]
	Matching rules: similar_ssn && similar_fname && similar_lname && similar_stadd && similar_city && similar_zip		

Table 5.14: True positives (tp) and false positives (fp) captured by each rule in Rule-based, Rule+EM(1), and Rule+EM(2) for DBGen dataset (50k records, 40065 duplicates). Refer to Appendix for the rulesets.

Approach	No.	Rule	tp	fp
Rule-based	1	similar_ssn && similar_name	20836	0
	2	(similar_ssn || similar_name) && (very_similar_address_1 || very_similar_address_2)	33136	0
	3	similar_ssn && similar_stadd && !similar_name	10026	0
	4	similar_ssn && very_similar_address_2 && similar_fname && similar_zip	21233	0
Rule+EM(1)	1	similar_ssn && similar_name	26435	0
	2	(similar_ssn || similar_name) && (very_similar_address_1 || very_similar_address_2)	37796	0
	3	similar_ssn && similar_stadd && !similar_name	5259	0
	4	similar_ssn && very_similar_address_2 && similar_fname && similar_zip	25861	0
Rule+EM(2)	1	similar_ssn && similar_fname && similar_lname	26435	0
	2	similar_city && similar_zip	38271	113

Table 5.14 shows the true positives and false positives obtained by each rule in the Rule-based approach, Rule+EM(1), and Rule+EM(2). The Equational Theory rules for the Rule-based approach is similar to the rules created by Hernandez et. al [7]. Rule+EM(2) consists of only two matching rules, both of which were able to identify many duplicates from the DBGen dataset. The second rule from Rule+EM(2) shows that attributes 'city' and 'zip' are good enough to discover matching record pairs, although more record pairs were wrongly classified as matching. Meanwhile, it is uncertain how Hernandez et. al. created the rules in the first place whereas the rules generated from Rule+EM(2) are based on statistics obtained from analysing the behaviour of the dataset itself. Thus, we can provide to a certain degree of confidence, the validity of the rules created from the Rule+EM(2) technique.

The results in Table 5.14 were computed using the first three SQL statements in Table 5.15 for the multi-pass SN method. These three candidate keys were composed of the most significant attributes as the first elements. From Table 5.12, it is obvious that

Table 5.15: True positives (tp) and false positives (fp) captured by candidate keys created from significant and insignificant attributes for Rule+EM(2) multi-pass SN method for DBGen dataset (50k records, 10k clusters).

No.	**SQL to generate candidate key**	**tp**	**fp**
1	UPDATE Cora SET candKey=UPPER(CONCAT(SUBSTRING(lname, 1, 3), SUBSTRING(fname, 1, 3), SUBSTRING(stadd, 1, 3)))	35989	43
2	UPDATE Cora SET candKey=UPPER(CONCAT(SUBSTRING(fname, 1, 3), SUBSTRING(zip, 1, 3), SUBSTRING(city, 1, 3)))	35907	70
3	UPDATE Cora SET candKey=UPPER(CONCAT(apmt, state, SUBSTRING(fname, 1, 3)))	35583	112
4	UPDATE Cora SET candKey=UPPER(CONCAT(SUBSTRING(ssn, 1, 3), SUBSTRING(stadd, 1, 3), SUBSTRING(zip, 1, 3)))	32014	99
5	UPDATE Cora SET candKey=UPPER(CONCAT(minit, stnum, SUBSTRING(city, 1, 3)))	29888	106
6	UPDATE Cora SET candKey=UPPER(CONCAT(stnum, minit, state))	28380	127

'fname' and 'lname' are the two most consistent non-trivial attributes with the highest m_i and the lowest u_i. Consequently, they are important as the first elements in the candidate keys. Meanwhile, attribute 'ssn' records a high weight as well but its value of m_i is not as high as 'apmt'. Table 5.15 reveals that candidate key with 'apmt' as the first element detected more duplicates than candidate key with 'ssn' as the first element, although at a slightly higher expense of falsely classifying non-matching pairs as duplicates. Finally, we show in the fifth and sixth SQL statements that trivial attributes such as 'state', 'stnum', and 'minit' are the most non-contributional attributes to increase the chance of duplicates falling into the sliding window.

Figure 5.5 compares the accuracy between the Rule-based, Rule+EM(1), Rule+EM(2), and the Probabilistic-based approaches for increasing dataset size up to 500k records. Figure 5.6 shows the F-scores for DBGen dataset of size 500k records. It is clearly shown that Rule+EM(2) with only two matching rules, outperforms the Rule-based approach,

Rule+EM(1), and the Probabilistic-based approach, with a recall score at around 98%. Meanwhile, the Probabilistic-based approach performs better than the Rule-based approach. This is because the EM algorithm works well under the conditions where there are many duplicates in the dataset and the number of attributes are large enough to compensate for errors in other attributes of the dataset.

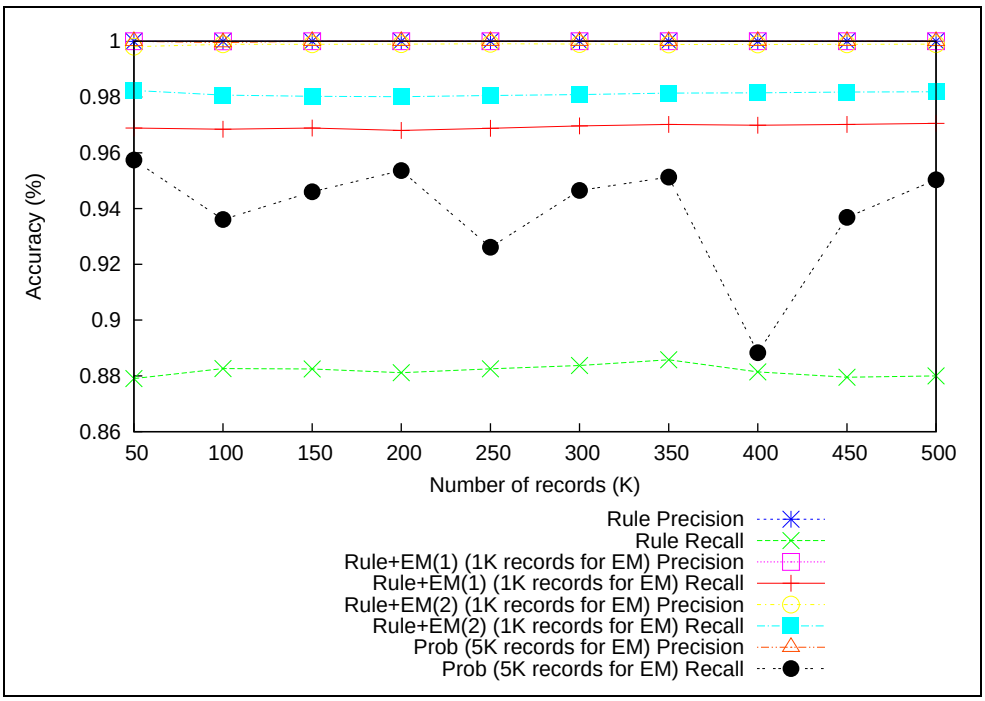

Figure 5.5: Accuracy comparison between Rule-based, Rule+EM(1), Rule+EM(2), and Probabilistic-based approach w.r.t. record size for DBGen dataset.

Finally, for the experiments resulting in Figure 5.7, we measured the scalability of the Rule-based approach, Rule+EM(1) and Rule+EM(2), and the Probabilistic-based approach. Note that the runtime displayed by Rule+EM(1), Rule+EM(2), and the Probabilistic-based approach refers only to the runtime during the matching process. The execution of the EM algorithm, which took around 26 seconds, is not included in the runtime as it was only run once to generate required information for Rule+EM(1), Rule+EM(2), and the Probabilistic-based approach. It is also obvious from the figure that the Rule-based approach has better efficiency than the Probabilistic-based approach.

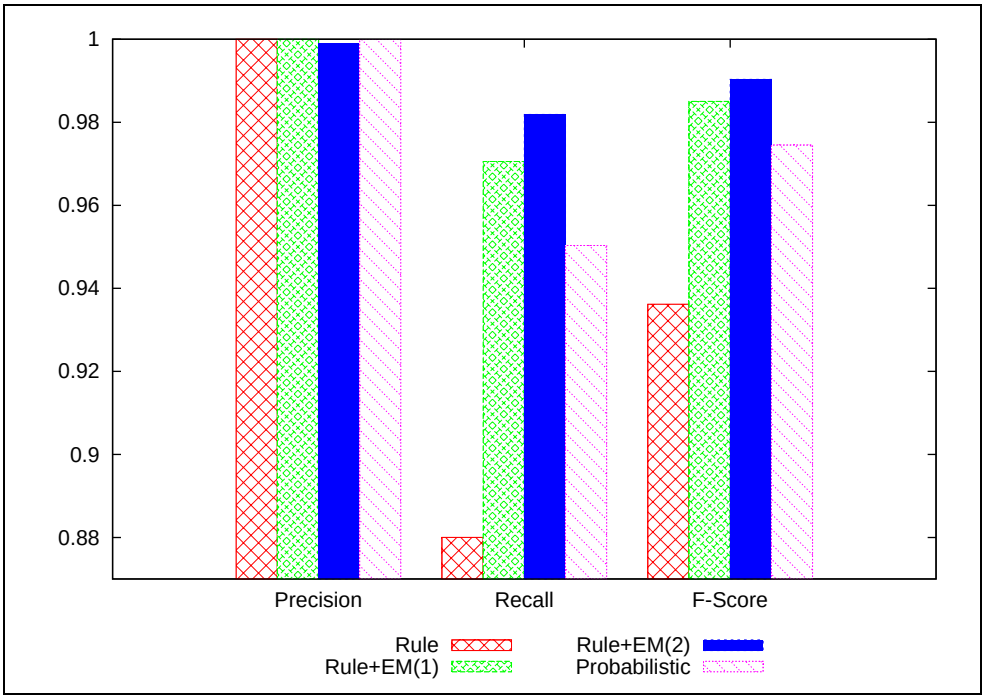

Figure 5.6: Accuracy comparison between Rule-based, Rule+EM(1), Rule+EM(2), and Probabilistic-based approach for DBGen dataset (500k records) when sliding window size is 10.

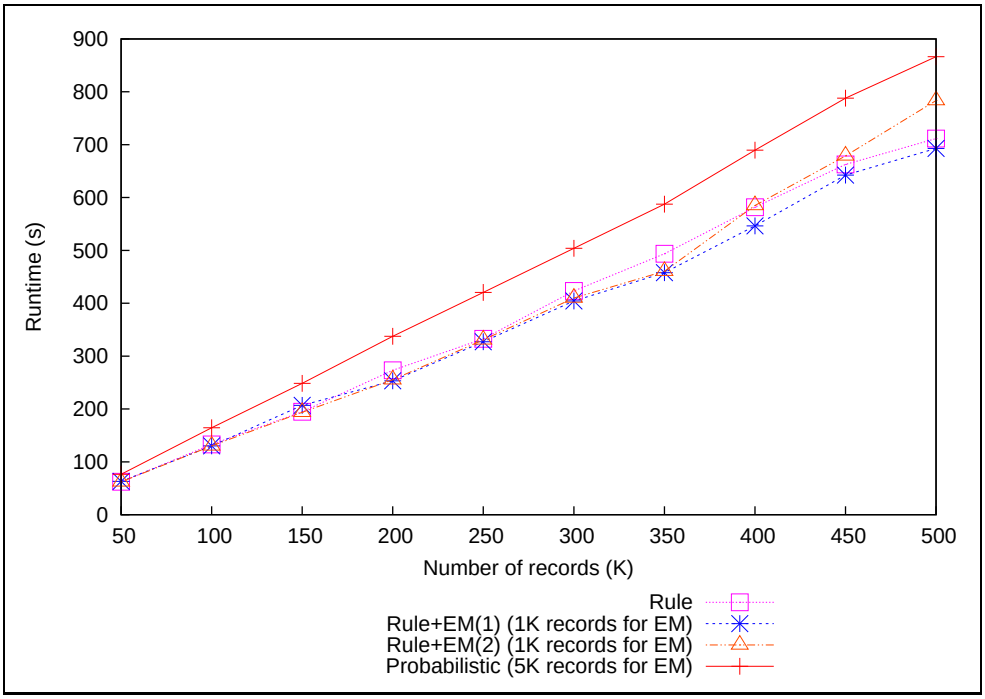

Figure 5.7: Runtime comparison between Rule-based, Rule+EM(1), Rule+EM(2), and Probabilistic-based approach w.r.t. record size for DBGen dataset.

5.3 Comparisons with Other Works

In this section, we make a comparison between the results we obtained in this thesis with other works that used similar datasets.

The first dataset in this project, the Restaurant dataset, was also used by Tejada et. al. [30]. They developed a learning object identification rules system called Active Atlas, which is similar to our work in the sense that it also determines mappings between objects. Similarly, the system determines which combination of attributes are important to decide the mapping between two objects. However, whilst we use the EM algorithm to help us decide the significant attributes, they proposed a measurement that determines the unique weights of individual attributes from training data. The rules were obtained via decision tree learning based on the computed weights and user input. The results obtained from this approach are 109 true positives and 2 false positives out of 112 duplicates. Meanwhile, our results using the Cocktail approach are 104 true positives and 3 false positives out of 112 duplicates. Our result is slightly worse but at the advantages of not requiring training data and user input.

Another work that used the Restaurant dataset is the work by Ravikumar and Cohen [31]. Their approach is also based on unsupervised method, similar to ours. Their proposed approach is based on a hierarchical graphical model to learn to match record pairs. However, their results were not as favourable, where their best result is only at 82% for precision and 84.4% for F-score. Our precision for the Cocktail approach is 97.2% whilst F-score is 95%.

Cohen and Richman [3] also used the Restaurant and Cora datasets in their research. Theirs is a supervised approach where the system learns from training data how to cluster records that refer to the same real-world entity in an adaptive way. The results they obtained for the Restaurant dataset is the best insofar, at 100% for both precision and recall. However, the results for the Cora dataset is slightly lower than ours, with an F-score at 96.4% (99% precision and 94% recall) whilst our best F-score for the Cora dataset is 98.52% (98.56% precision and 98.48% recall).

Singla and Domingos [32] proposed an approach that also used the Cora dataset to solve the record matching problem. The approach they have taken is using a collective technique to propagate information among related decisions via common attributes between records. Their best result is at 89% F-score (84.5% precision and 94.9% recall),

which is also lower than our results for the Cora dataset.

Last of all, we compare our results for the DBGen dataset with the original work by Hernandez et. al. [7]. It is shown in Figure 2 of their paper that the recall percentage for a DBGen dataset of 1 million records (with 423644 duplicates) is around 88%. We used the similar rules created by Hernandez et. al. (refer to Algorithm 15 in the Appendix) for our baseline approach, which is the Rule-based approach. Our results in Figure 5.5 show that we also obtained about 88% recall when using the similar rules, for varying DBGen dataset size up to 500k records. Although we were not able to grow our database to 1M records, we can expect a recall at 88% as well, from the analysis of our results in Figure 5.5. The Cocktail approach proves to be a major improvement compared to the baseline result (based on the rules created by Hernandez et. al.), where we managed to obtain around 99% recall and 98% precision.

Overally, the results we have obtained from the Cocktail approach are favourable. The Cocktail approach also has added advantages, in the sense that it does not require training data and user input, unlike some of the approaches described in this section.

Chapter 6

Conclusion And Future Work

The biggest challenge of this project is the research to improve the current approaches to record matching. The improvements could have been in terms of accuracy, automation versus manual effort, or time performance. Upon our research, we decided to work on the Rule-based approach and Probabilistic-based approach because they have been heavily researched for the past couple of years and are popular techniques in most of today's available record matching softwares. We were especially interested in the Rule-based approach, especially on the design of a good matching ruleset. We have found that, statistics from the EM algorithm can be used to guide us to achieve this, as well as to automate the process of the rules creation. Meanwhile, the second challenge of the project is implementing a good system design. Since we were dealing with large datasets, careful manipulation and querying of the datasets were imperative to avoid causing the system to run slow.

The experiments we have conducted proved that the Probabilistic-based approach (with the EM algorithm) works better than the Rule-based approach when the number of duplicates in the dataset is large enough to accurately compute the maximum likelihood estimate. The Rule-based approach did not perform well because the rulesets manually created were not as effective as we have hoped for. This is because the rulesets were devised without knowledge of the data behaviour i.e. if an attribute has many missing or erroneous data. The Cocktail approach has the best Recall and F-scores percentages for all the datasets. This proves that the Rule-based approach works best when the matching ruleset is accurate and effective.

The success of the Cocktail approach in this project is predominantly determined by

the success of the EM algorithm. The EM algorithm works well for the Restaurant, Cora, and DBGen datasets because there are more than 10% duplicates in each dataset. For cases where the requirements in [15] are not met, we suggest the use of training data to estimate the EM values to automatically derive the matching ruleset.

In summary, the EM algorithm is a major contributor to the Cocktail approach because we were able to determine which attributes were the most significant in the datasets from the weight scores of the attributes. We proved that the accuracy of the record matching approach improved when significant attributes were used in the candidate keys for subspaces creation. Meanwhile, by ignoring insignificant attributes in the creation of Equational Theory ruleset, we were able to obtain better results as well. Insignificant attributes in the experiments, were largely due to the high values of both m_i and u_i, which showed that the attributes always match even for non-duplicates, or low values of both m_i and u_i, which showed that the data for the attribute were largely incomplete or often incorrect. In the experiments, we looked for the significant and insignificant attributes using a few blocking keys. Attributes that have been found to be insignificant in all cases were not used for matching rules creation. On the other hand, attributes that have been found to be significant in all cases, or significant in some but not in a few other cases, were considered as potential attributes to create a mapping (matching rule) between record pairs that agree on these attributes.

One possible direction to improve the accuracy of the Cocktail approach is by considering non-binary values in the vector patterns. In this project, we have only considered binary values where the attribute of a vector pattern is given a '1' if the attribute similarity score between a record pair satisfies the threshold value at 0.8, and a '0' if it does not. There is certainly room for improvements by regulating the threshold value to create non-binary vector patterns, hence matching rules of varying strictness level.

There are also other works that can be similarised to our work in this project. The novel approach by Fan and Jia [29] is also based on the Rule-based approach. They improved upon the Rule-based approach by introducing heterogeneous functional dependencies (HFDs) for specifying and reasoning about matching rules. The HFDs is an extension of the traditional functional dependencies, which can capture correspondences between attributes across multiple heterogeneous relations, and reason about both equality and similarity.

Another work, which also involves reasoning technique to automatically derive match-

ing rules, is the work by Tejada et. al. [30]. The system employed decision trees to teach rules, through limited user input, for a specific application domain. Similar to this project, they also compared between attribute weights to decide which attributes are important to create a mapping between two objects. While their attribute weights were measured by the total number of unique attribute values contained in the attribute set divided by the total number of values for the set, our attribute weights were computed from the EM algorithm.

An interesting direction for future research is to develop techniques that combine useful features from our Cocktail approach with the mentioned two approaches.

Appendix A

Equational Theory Rules

Input: Similarity score for each attributes in record pair A and B

Output: Record pair A and B is matching or non-matching

```
1  if similar_addr AND similar_city then
2      very_similar_address = true;
3  if phone_similarity is 1 then
4      very_similar_phone = true;
5  if very_similar_phone AND (similar_name OR very_similar_address) then
6      match = true;
7  else if similar_name AND very_similar_addr AND similar_phone then
8      match = true;
9  else if similar_name AND similar_phone AND similar_type then
10     match = true;
11 else
12     match = false;
```

Algorithm 15: Original ruleset for Restaurant dataset without the influence of EM algorithm

Input: Similarity score for each attributes in record pair A and B

Output: Record pair A and B is matching or non-matching

```
1 if phone_similarity is 1 then
2     very_similar_phone = true;
3 if very_similar_phone AND (similar_name OR similar_address) then
4     match = true;
5 else if similar_name AND similar_phone then
6     match = true;
7 else
8     match = false;
```

Algorithm 16: Rule+EM(1) ruleset for Restaurant dataset after removing insignificant attributes

Input: Similarity score for each attributes in record pair A and B

Output: Record pair A and B is matching or non-matching

```
1 if similar_name AND similar_phone then
2     match = true;
3 else
4     match = false;
```

Algorithm 17: Rule+EM(2) ruleset for Restaurant dataset, automatically derived from EM algorithm

Input: Similarity score for each attributes in record pair A and B

Output: Record pair A and B is matching or non-matching

```
1  if similar_month AND similar_year then
2      similar_date;
3  if similar_author AND similar_title AND similar_year then
4      match = true;
5  else if similar_title AND similar_venue AND similar_year then
6      match = true;
7  else if similar_author AND similar_venue AND similar_date then
8      match = true;
9  else if (similar_author OR similar_title) AND similar_pages AND similar_volume AND
   _publisher then
10     match = true;
11 else if similar_title AND similar_date then
12     match = true;
13 else
14     match = false;
```

Algorithm 18: Original ruleset for Cora dataset without the influence of EM algorithm

Input: Similarity score for each attributes in record pair A and B

Output: Record pair A and B is matching or non-matching

```
1  if similar_author AND similar_title then
2      match = true;
3  else if similar_title AND (similar_venue OR similar_month) then
4      match = true;
5  else if similar_author AND similar_venue AND similar_month then
6      match = true;
7  else if (similar_author OR similar_title) AND similar_pages AND similar_volume AND
     _publisher then
8      match = true;
9  else
10     match = false;
```

Algorithm 19: Rule+EM(1) ruleset for Cora dataset after removing insignificant attributes

Input: Similarity score for each attributes in record pair A and B

Output: Record pair A and B is matching or non-matching

```
1  if similar_author AND similar_title then
2      match = true;
3  else if (similar_title OR similar_volume) AND similar_pages then
4      match = true;
5  else if similar_publisher then
6      match = true;
7  else if similar_venue then
8      match = true;
9  else if similar_address then
10     match = true;
11 else
12     match = false;
```

Algorithm 20: Rule+EM(2) ruleset for Cora dataset, automatically derived from EM algorithm

Input: Similarity score for each attributes in record pair A and B

Output: Record pair A and B is matching or non-matching

```
1  if similar_fname AND similar_minit AND similar_lname then
2      similar_name;
3  if similar_stadd AND similar_city AND (similar_state OR similar_zip) then
4      very_similar_address_1;
5  if (similar_stnum AND similar_aptm AND similar_city AND (similar_state OR
   similar_zip) AND (!similar_stAdd)) OR (similar_stAdd AND similar_stnum AND
   similar_aptm AND similar_zip) then
6      very_similar_address_2;
7  if similar_ssn AND similar_name then
8      match = true;
9  else if (similar_ssn OR similar_name) AND (very_similar_address_1 OR
   very_similar_address_2) then
10     match = true;
11 else if similar_ssn AND similar_stAdd AND !similar_name then
12     match = true;
13 else if similar_ssn AND very_similar_address_2 AND similar_fname AND similar_zip
   then
14     match = true;
15 else
16     match = false;
```

Algorithm 21: Original ruleset for DBGen dataset without the influence of EM algorithm

```
Input: Similarity score for each attributes in record pair A and B
Output: Record pair A and B is matching or non-matching
1  if similar_fname AND similar_lname then
2      similar_name;
3  if similar_stadd AND similar_city then
4      very_similar_address_1;
5  if (similar_city AND !similar_stAdd) OR (similar_stAdd AND similar_zip) then
6      very_similar_address_2;
7  if similar_ssn AND similar_name then
8      match = true;
9  else if (similar_ssn OR similar_name) AND (very_similar_address_1 OR
   very_similar_address_2) then
10     match = true;
11 else if similar_ssn AND similar_stAdd AND !similar_name then
12     match = true;
13 else if similar_ssn AND very_similar_address_2 AND similar_fname AND similar_zip
   then
14     match = true;
15 else
16     match = false;
```

Algorithm 22: Rule+EM(1) ruleset for DBGen dataset after removing insignificant attributes

```
Input: Similarity score for each attributes in record pair A and B
Output: Record pair A and B is matching or non-matching
1  if similar_ssn AND similar_fname AND similar_lname then
2      match = true;
3  else if similar_city AND similar_zip then
4      match = true;
5  else
6      match = false;
```

Algorithm 23: Rule+EM(2) ruleset for DBGen dataset, automatically derived from EM algorithm

Bibliography

[1] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64:1183–1210, 1969.

[2] T. Joachims. Making large-scale svm learning practical. In MIT Press, editor, *Advances in Kernel Methods-Support Vector Learning*, 1999.

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

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

[5] P. Singla and P. Domingos. Multi-relational record linkage. In *Proc. KDD-2004 Workshop Multi-Relational Data Mining*, 2004.

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

[7] M. A. Hernandez and S. J. Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. In *Data Mining and Knowledge Discovery*, 1998.

[8] E.-P. Lim, J. Srivastava, and J. Richardson. Entity identification in database integration. In *Proc. Ninth IEEE Int'l Conf. Data Eng. (ICDE '93)*, pages 294–301, 1993.

[9] H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C. A. Saita. Declarative data cleaning: Language, model, and algorithms. In *Proc. 27th Int'l Conf. Very Large Databases (VLDB '01)*, 2001.

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

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

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

[13] P. Christen and T. Churches. Febrl: Freely extensible biomedical record linkage manual.

[14] M. G. Elfeky, A. K. Elmagarmid, and V. S. Verykios. Tailor: A record linkage tool box. In *Proc. 18th IEEE Int'l Conf. Data Eng. (ICDE '02)*, 2002.

[15] W. E. Winkler. Methods for record linkage and bayesian networks. In *Technical Report Statistical Research Report Series RRS2002/05, US Bureau of the Census, Washington, D.C.*, 2002.

[16] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):1–16, 2007.

[17] M. A. Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *Journal of the American Statistical Association*, 89:414–420, 1989.

[18] Rohan Baxter, Peter Christen, and Tim Churches. A comparison of fast blocking methods for record linkage. In *In Proceedings of 9th ACM SIGKDD Workshop on Data Cleaning, Record Linkage and Object Consolidation*, 2003.

[19] William W. Cohen, Pradeep Ravikumar, and Stephen E. Fienberg. A comparison of string metrics for matching names and records. In *Data Cleaning Workshop in Conjunction with KDD*, 2003.

[20] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. In *J. Molecular Biology*, 1970.

[21] P Jaccard. The distribution of flora in the the distribution of flora in the alpine zone. In *The New Phytologist*, 1912.

[22] H. B. Newcombe, J. M. Kennedy, S. J. Axford, and A. P. James. Automatic linkage of vital records. *Science*, 130:954–959, October 1959.

[23] William E. Winkler and Yves Thibaudeau. An application of the fellegi-sunter model of record linkage to the 1990 u.s. decennial census. Technical report, U.S. Bureau of the Census, Statistical Research Division Technical report RR91/09, 1991.

[24] N.M. A.P., Laird and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society*, 39:1–38, 1977.

[25] W. E. Winkler. Improved decision rules in the felligi-sunter model of record linkage. In *Technical Report Statistical Research Report Series RR93/12, US Bureau of the Census, Washington*, 1993.

[26] Y. R. Wang and S. E. Madnick. The inter-database instance identification problem in integrating autonomous systems. In *Proc. fifth IEEE Int'l Conf. data Eng. (ICDE '89)*, 1989.

[27] Mikhail Bilenko and Raymond J. Mooney. On evaluation and training-set construction for duplicate detection on evaluation and training-set construction for duplicate detection on evaluation and training-set construction for duplicate detection on evaluation and training-set construction for duplicate detection. In *Proceedings of the KDD-2003 Workshop on Data Cleaning, Record Linkage, and*, 2003.

[28] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schutze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[29] X. Jia W. Fan. *Record Matching with Heterogeneous Functional Dependencies*. University of Edinburgh.

[30] S. Tejada, C. A. Knoblock, and S. Minton. Learning object identification rules for information integration. *Information Systems*, 26(8):607–633, 2001.

[31] P. Ravikumar and W. W. Cohen. A hierarchical graphical model for record linkage. In *20th Conf. Uncertainty in Artificial Intelligence (UAI '04)*, 2004.

[32] Parag Singla and Pedro Domingos. Object identification with attribute-mediated dependences. In *In Conference on Principals and Practice of Knowledge Discovery in Databases (PKDD)*, pages 297–308, 2005.

[33] A. E. Monge and C. P. Elkan. An efficient domain-independent algorithm for detecting approximately duplicate database records. *Proc. Second ACM SIGMOD Workshop Research Issues in Data Mining and Knowledge Discovery (DMKD '97)*, (23-29), 1997.