COMBINATION OF A PROBABILISTIC-BASED AND A RULE- BASED APPROACH FOR GENEALOGICAL RECORD LINKAGE

A Thesis

presented to

the Faculty of California Polytechnic State University

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Pooja Shah

February 2015

COMMITTEE MEMBERSHIP

TITLE:	Combination of a Probabilistic-Based and a Rule-Based Approach for Genealogical Record Linkage
AUTHOR:	Pooja Shah
DATE SUBMITTED:	February 2015
COMMITTEE CHAIR:	Franz Kurfess, Ph.D. Professor of Computer Science
COMMITTEE MEMBER:	Foaad Khosmood, Ph.D. Assistant Professor of Computer Science
COMMITTEE MEMBER:	Hisham Assal, Ph.D. Lecturer of Computer Science
COMMITTEE MEMBER:	Brian Leverich, Ph.D. Owner, Linkpendium

ABSTRACT

Combination of a Probabilistic-Based and a Rule-Based Approach for Genealogical Record Linkage

Pooja Shah

Record linkage is the task of identifying records within one or multiple databases that refer to the same entity. Currently, there exist many different approaches for record linkage. Some approaches incorporate the use of heuristic rules, mathematical models, Markov models, or machine learning. This thesis focuses on the application of record linkage to genealogical records within family trees. Today, large collections of genealogical records are stored in databases, which may contain multiple records that refer to a single individual. Resolving duplicate genealogical records can extend our knowledge on who has lived and more complete information can be constructed by combining all information referring to an individual. Simple string matching is not a feasible option for identifying duplicate records due to inconsistencies such as typographical errors, data entry errors, and missing data.

Record linkage algorithms can be classified under two broad categories, a rule-based or heuristic approach, or a probabilistic-based approach. The Cocktail Approach, presented by Shirley Ong Ai Pei, combines a probabilistic-based approach with a rule-based approach for record linkage. This thesis discusses a re-implementation and adoption of the Cocktail Approach to genealogical records.

ACKNOWLEDGMENTS

I would like to thank Dr. Franz Kurfess and Dr. Brian Leverich for their advice and guidance throughout this thesis. I would also like to thank Dr. Foaad Khosmood and Dr. Hisham Assal for their feedback as it helped me to improve my thesis.

The dataset used in this thesis was provided by Western Michigan Genealogical Society from its collection and made available by Roger Moffat of WMGS.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

The amount of data being stored in databases is increasing. Large databases of records may contain multiple records that refer to the same entity, and each of these records may contain identical, similar, or unique information. Therefore it is important for computers to have an efficient and accurate method of automatically detecting two records that refer to the same entity. Record linkage is the task of identifying records in one or multiple databases that refer to the same entity. It is used to find and remove duplicates, or merge records together. Record attributes and common identifiers are used to identify matches/duplicates [23]. Record linkage is applicable to many different domains, including genealogy. With an increased interest in family history research, the need for improvement in genealogical record linkage has risen [13]. Today, large collections of genealogical data are stored in databases, allowing genealogists to search large quantities of data instantaneously. Computers have greatly helped to gather and organize genealogical information. This thesis focuses on the application of record linkage to genealogical records within family trees. By finding duplicate records, previously unknown links between different family trees may be discovered, allowing the trees to be merged together. In addition, more complete information can be constructed by combining all records referring to a sole individual [13].

Information from genealogical records is used to build family trees that represent family relationships. Each record has several attributes that provide

additional information about an individual. Information about family trees is usually stored in Genealogical Data Communication (GEDCOM) files, an industry standard data format for genealogical information [4]. GEDCOM files uses a standard ASCII format and contain metadata linking the records together. It was developed for exchanging genealogical data among different genealogy software. Several different software products are available on the market to help one track their family history, build trees, and find duplicates records. Unfortunately, many software products suffer from weak merging algorithms and are error prone due to slight differences in records, making collaboration difficult. One technique is to assign unique identifiers to each individual however, unique identifiers are not support by or standardized among all software products.

In the past, genealogical record linkage has been done manually, which is very time consuming. Automating the process of identifying duplicate records reduces the burden on users and genealogists when merging family trees [21]. Computers have the advantage of better quality control, speed, consistency, reproducibility of results, and significant reduction of human error [23]. It also makes it easier for genealogists to collaborate during their research. The hardest part of automating the process is finding all the duplicate records. Genealogical record linkage is a hard problem due to many inconsistencies among the data and common naming conventions within a society and/or family.

Records about an individual that are spread across different genealogical databases may not be identical due to human error such as misspellings or data entry errors. Records may also have missing data, lexical heterogeneity, or

structural heterogeneity. Structural heterogeneity refers to records with different domain structures. For example, an individual's full address may be stored in the 'address' attribute in one record, but stored in multiple different attributes, such as 'city', 'state', and 'zip code' in another record. Structural heterogeneity will not be addressed in this thesis. Lexical heterogeneity refers to records with similar structure but different representations of data, such as 'A. Hall' and 'Ann Hall' [17]. Table 1.1 shows three non-matching records that refer to the same individual. These records contain missing fields, misspelled names, or data entry errors. The last name in the third record is different from the other two records and may be due to a name change. These are all common problems found within genealogical records. Due to inconsistencies between records, simple string matching is not a feasible option for genealogical record linkage. A genealogist may be able to compare these records and conclude that they refer to the same individual. However, it is much harder for a computer to make these conclusions.

First Name	**Middle Name**	**Last Name**	**Birth Date**	**Death Date**
Ann	Mae	Hall		26 SEP 1954
Ann		Hal	18 MAY 1902	
Ann	M.	Smith	18 MAY 1902	20 SEP 1954

Table 1.1 Three non-matching records that refer to the same individual.

In order for a genealogist to identify duplicate records, he/she needs to carefully analyze the dataset to understand how the records behave and

understand the domain. For example, early civil and church records may use different spellings of names in different records of the same individual. Nicknames may be used, dates may be misreported, or day and month may be interchanged [13]. Genealogical records are very different for each region and time period. The goal of this thesis is to find an efficient record linkage algorithm that reduces the need of human intervention and has a high accuracy of finding duplicate genealogical records within family trees. The Cocktail Approach is a general record matching approach presented by Shirley Ong Ai Pei that yielded high recall, precision, and f-scores [17]. The Cocktail Approach uses statistics about a dataset to generate a ruleset to find duplicate records. This thesis discusses a re-implementation and adoption of the Cocktail Approach to genealogical records. My contributions include:

- The adaption and evaluation of the Cocktail Approach in the genealogical domain.
- Modifying the Cocktail Approach to also analyze relationships among individuals.
- Incorporating general domain metarules to the generated ruleset.

Chapter 2 reviews related work and Chapter 3 explains the mathematical model used in this thesis. Chapter 4 presents additional components needed in record linkage and Chapter 5 give a high level overview of the Cocktail Approach. Chapter 6 give an overview of my implementation of the Cocktail Approach and Chapter 7 presents the testing results and evaluation. Finally, Chapter 8 concludes the thesis and Chapter 9 discusses future work.

CHAPTER 2

Related Work

There has been plenty of research into record linkage, which is also sometimes called object identification [20], data cleaning [19], approximate matching [9], fuzzy matching [1], or entity resolution [3]. Many different approaches have been presented for solving the problem. Tejada presents an approach that learns to simultaneously create both mapping rules and a set of general transformations to a specific application domain through limited user input [20]. Winkler demonstrated how Markov Chains and ideas from Fellegi and Sunter could be used to improve the learnability of Bayesian Networks. Winkler also used ideas from machine learning to show how record linkage classifiers can make better use of training data [24]. Other researches have incorporated techniques from machine learning, data mining, and artificial intelligence. Many of these approaches are based on supervised learning techniques and require training data [8]. However, training data may not always be available in real world situations due to privacy concerns. Alternatively, some approaches do not require training data to achieve high accuracies. The Cocktail Approach is one such approach. This approach makes use of statistics obtained from the dataset to gain a better understanding of the behavior of the data and to generate a ruleset used to match duplicate records, or matching ruleset.

Although there exist several approaches for record linkage, most approaches are not built for any specific domain. Records in different domains may behave differently. Existing algorithms/systems may not work optimally on

records having characteristics significantly different from those on which it was developed [23]. Therefore record linkage algorithms will have to be slightly altered for each domain. This thesis focuses specifically on genealogical record linkage.

Genealogical records tend to have several attributes that describe an individual such as: first name, last name, middle name, address, birth and death dates, and relatives. One technique for genealogical record linkage is to first assign weights (positive and negative) to each attribute to account for missing entries. A training set of records must be used to estimate the weights for each attribute [13]. When comparing two records, the positive weight for an attribute is added to the total sum if the attributes match, and the negative weight is added if the attributes do not match. A zero weight is used if the attribute is blank in one or both records. The total sum, or score, for each record pair is compared against a predefined threshold to determine if the records match or not.

Many record linkage algorithms can be classified under two broad categories: a rule-based approach and a probabilistic-based approach. Matching rulesets are created by expert genealogists through substantial effort and time, and tend to obtain high accuracies [17]. The probabilistic approach does not need substantial human effort, but it relies on training data to compute the maximum likelihood to determine whether or not two records match.

2.1 Rule-Based Approach

Rule-based approaches use a set of heuristic rules to determine if two records match or not. They have the potential of reaching high accuracies because they

capture the heuristic knowledge from experts while probabilistic-based approaches capture casual dependencies based on statistics. The weakness with a rule-based approach lies in the manual tuning of rules, which requires a deep understanding and analysis of the domain and behavior of the data itself [17]. In addition, a matching ruleset may not be applicable to a pair a records, or dataset, that are different than those used in defining the ruleset [23]. Behaviors of genealogical records are significantly different for each region and time period. Therefore a new ruleset would have to be created for each dataset.

2.1.1 Equational Theory Ruleset

Hernandez and Stolfo [10] stated that the fundamental problem of record linkage is that the data supplied by various sources typically include string data that are different among different datasets. They suggested that the equality of two values should be specified by an equational theory, or a set of equational axioms that define equivalence. An approach to defining an equational theory for practical uses is through a ruleset [10]. Figure 2.1 shows an example of an equational theory rule for finding duplicate records. The idea is to define an equational theory that dictates the logic of domain equivalence, and not just string equivalence [10]. Calculating similarities among strings can be done by some distance function. The score is compared against a predefined threshold value to account for any typographical errors in the data.

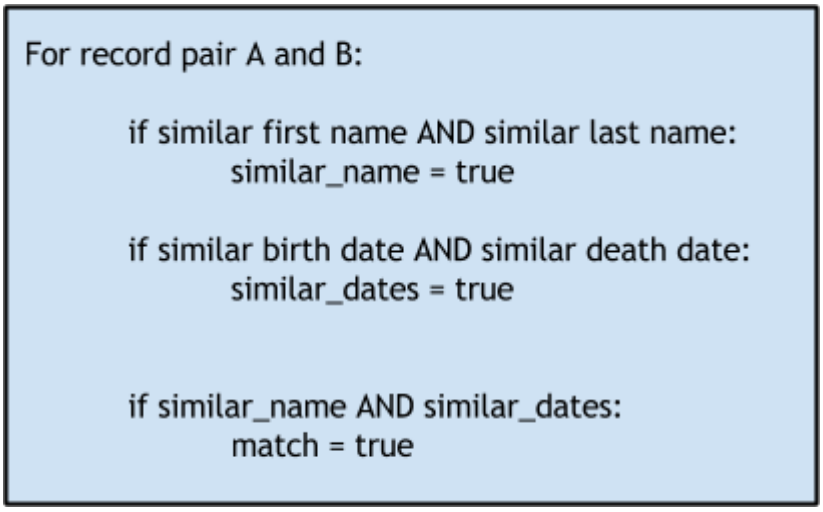

Figure 2.1 An example equational theory rule used to determine if two records, A and B, refer to the same individual.

2.2 Probabilistic-Based Approach

Probabilistic-based record linkage links records that are not necessarily identical but close in some fields. It is a way for a computer to mimic some of the decision-making processes a genealogist may use to recognize valid variations in the data [13]. Many probabilistic-based approaches need training data to receive high accuracy rates. However, it is possible for similar databases in the same domain to behave differently. Thus, using one database of records as training data may not give accurate information on how another database of records behave. An alternative method is to use unsupervised probabilistic methods such as the Expectation Maximization (EM) algorithm, which is a means of obtaining maximum likelihood estimates for incomplete data [17].

In 1969, Fellegi and Sunter [7] introduced a mathematical model based on the concepts by Newcombe [16]. The ideas of Fellegi and Sunter became the basis of modern probabilistic record linkage because they introduced many ways

of computing key parameters needed for the matching process [23]. The Fellegi-Sunter model and the EM algorithm are discussed in Chapter 3.

2.3 Graph-Based Remerging of Genealogical Databases

Dr. Randall Wilson presented an algorithm that takes into account the relationships of individuals in a database when merging two genealogical databases together. The algorithm first attempts to identify all individuals that appear in both databases that can be assumed with a high probability to be matches. The algorithm then presents any conflicts to the user so they can incorporate new data into the database. The key to this algorithm is to use the relationships among individuals to increase the confidence in matches. The idea is that it is unlikely that two different records will have the same relationship to dozens of other people unless they actually refer to the same individual [21].

CHAPTER 3

Fellegi-Sunter Model and the EM Algorithm

3.1 Fellegi-Sunter Model

Modern probabilistic record linkage began with the work from Newcombe [16]. The Fellegi-Sunter model was created in 1969 based off the ideas of Newcombe, and is now the basis for modern record linkage. The idea is to classify pairs in a product space, $A \times B$, from two datasets, A and B, into a set of true matches, M, and a set of true non-matches, U [26]. The set of ordered pairs is represented by:

$$A \times B = \{(a, b); a \in A, b \in B\} \tag{3.1}$$

The set of pairs is the union of two disjoint sets:

$$M = \{(a, b); a = b, a \in A, b \in B\} \tag{3.2}$$

and

$$U = \{(a, b); a \neq b,\ a \in A, b \in B\} \tag{3.3}$$

Record pairs are represented by comparison vectors,

$$\gamma[\alpha(a), \beta(b)] = \{\gamma^1[\alpha(a), \beta(b)], \ldots, \gamma^K[\alpha(a), \beta(b)]\} \tag{3.4}$$

where $\alpha(a)$ correspond to members of set A and $\beta(b)$ correspond to members of set B [17]. In the comparison vectors, $\gamma^i,\ i = 1,\ldots,K$ represents a specific comparison between the same $i\text{-}th$ attribute of record a and record b. Γ is defined as the comparison subspace that represents the set of all possible vectors

patterns γ. A linkage rule, L, is defined as a mapping from Γ onto a set of random decision functions $D = \{d(\gamma)\}$ [17].

$$d(\gamma) = \{P(A_1|\gamma), P(A_2|\gamma), P(A_3|\gamma)\};\ \gamma \in \Gamma \tag{3.5}$$

and

$$\sum_{i=1}^{3} P(A_i|\gamma) = 1 \tag{3.6}$$

A record pair is considered a match if the probability that it is a match, $P(M|\gamma[\alpha(a), \beta(b)])$, is greater that the probability that it is a non-match, $P(U|\gamma[\alpha(a),\beta(b)])$. Bayes decision rule for minimum error is used to determine the probabilities. These values can be computed using a training set of pre-labeled record pairs or using the EM algorithm. The likelihood ratio is defined as

$$R = R[\gamma(a,b)] = \frac{m(\gamma)}{u(\gamma)} \tag{3.7}$$

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

$$m(\gamma) = P\{\gamma[\alpha(a), \beta(b)]|(a,b) \in M\} = \sum_{(a,b)\in M} P\{\gamma[\alpha(a), \beta(b)]\} \cdot P[(a,b)|M] \tag{3.8}$$

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

$$u(\gamma) = P\{\gamma[\alpha(a), \beta(b)]|(a,b) \in U\} = \sum_{(a,b)\in U} P\{\gamma[\alpha(a), \beta(b)]\} \cdot P[(a,b)|U] \tag{3.9}$$

The decision rule is given by [25]:

- If $R > T_\mu$, then designate the pair as a match.
- If $T_\lambda \leq R \leq T_\mu$, then designate the pair as a possible match and hold for clerical review.
- If $R < T_\lambda$, then designate the pair as a non-match.

The cutoff thresholds T_μ and T_λ are determined by a priori error bounds on false matches and false non-matches [25]. The idea behind this is if $\gamma \in \Gamma$ consists primarily of agreements, then it is intuitive that $\gamma \in \Gamma$ would be more likely to occur among matches than non-matches and the ratio would be large. On the other hand, if $\gamma \in \Gamma$ consists primarily of disagreements, then the ratio would be small [26].

3.2 Expectation Maximization Algorithm

The Expectation Maximization (EM) algorithm finds the maximum likelihood that different records refer to the same individual where data is incomplete. The EM algorithm is a good alternative in situations where training data is unavailable [17]. The algorithm is divided into two steps: The E-step and the M-step. The E-step calculates the expected likelihood that the records are a match and provides estimates for the missing data. The M-step sets the derivative of the data's log-likelihood to zero and updates the estimates of the unknown parameters.

The EM algorithm generates a parameter set $\phi=(m, u, p)$. The steps to generate ϕ are:

1. Give initial estimated values of ϕ.
2. Compute E-step using the values of ϕ.
3. Compute M-step to re-estimate the values ϕ using the values from the E-step.
4. Repeat steps 2 and 3 until the convergence of ϕ.

Both Pei [17] and Jaro [12] used a binary model for comparison vector γ such that $\gamma^j_i = 1$ if attribute i agrees for record pair j, and $\gamma^j_i = 0$ if attribute i disagrees for record pair j, for $i = 1, ..., n$ attributes and $j = 1, ..., N$ record pairs. The m_i and u_i probabilities can be defined as:

$$m_i = P\{\gamma^j_i = 1/r_j \in M\} \tag{3.10}$$

and

$$u_i = P\{\gamma^j_i = 1/r_j \in U\} \tag{3.11}$$

p is defined as the proportion of matched pairs.

$$p = \frac{|M|}{|M \cup U|} \tag{3.12}$$

Let x be the complete data vector equal to $\langle \gamma, g \rangle$, where $g_j = (1, 0)$ if and only if $r_j \in M$ and $g_j = (0, 1)$ if and only if $r_j \in U$. The complete data log-likelihood is [17, 22]:

$$ln\, f(x|\phi) = \sum_{j=1}^{N} g_j \cdot (ln\ P\{\gamma^j|M\}, ln\ P\{\gamma^j|U\})^T + \sum_{j=1}^{N} g_j \cdot (ln\, p, ln\,(1-p))^T \tag{3.13}$$

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

$$g_m(\gamma^j) = \frac{p \prod_{i=1}^{n} m_i^{\gamma_i^j} (1-m_i)^{1-\gamma_i^j}}{p \prod_{i=1}^{n} m_i^{\gamma_i^j} (1-m_i)^{1-\gamma_i^j} + (1-p) \prod_{i=1}^{n} u_i^{\gamma_i^j} (1-u_i)^{1-\gamma_i^j}} \tag{3.14}$$

$$g_u(\gamma^j) = \frac{(1-p) \prod_{i=1}^{n} u_i^{\gamma_i^j} (1-u_i)^{1-\gamma_i^j}}{p \prod_{i=1}^{n} m_i^{\gamma_i^j} (1-m_i)^{1-\gamma_i^j} + (1-p) \prod_{i=1}^{n} u_i^{\gamma_i^j} (1-u_i)^{1-\gamma_i^j}} \tag{3.15}$$

The values of $(g_m(\gamma^j), g_u(\gamma^j))$ are used in the M-step to calculate the values of ϕ. The equations are:

$$m_i = \frac{\sum_{j=1}^{N} \gamma_i^j \cdot g_m(\gamma^j)}{\sum_{N}^{j=1} g_m(\gamma^j)} \tag{3.16}$$

$$u_i = \frac{\sum_{j=1}^{N} \gamma_i^j \cdot g_u(\gamma^j)}{\sum_{N}^{j=1} g_u(\gamma^j)} \tag{3.17}$$

$$p = \frac{\sum_{j=1}^{N} g_m(\gamma^j)}{N} \tag{3.18}$$

After ϕ is generated, the weights of each attribute of a record pair are computed based on the values m and u. The score of each record pair is obtained by summing up all the weights of each attribute. Attributes that match make a positive contribution to the sum and attributes that do not match make a negative contribution. If the sum of the weights is greater than the upper threshold, T_μ, then the records are a match. If the sum is less than the lower threshold, T_λ, then the records are not a match. If the sum falls between the threshold values, then the results are non-conclusive. The upper and lower threshold values are defined from the Fellegi-Sunter model [17].

CHAPTER 4

Additional Components of Record Linkage

4.1 Comparison Subspaces

In several situations, databases are too large to consider every possible pair. Newcombe showed how to reduce the number of pairs considered by only considering pairs that agree on a particular characteristic [16, 26]. This is called blocking and is used to reduce the number of candidate record comparison pairs. The blocking approach attempts to find matching records by first sorting the records based on one or more attributes, or a candidate key. Once the records are sorted, they are separated into mutually exclusive partitions based on a blocking key or the candidate key. Once separated, only records within the same block will be considered for comparison. Figure 4.1 shows an example of how records are separated into blocks.

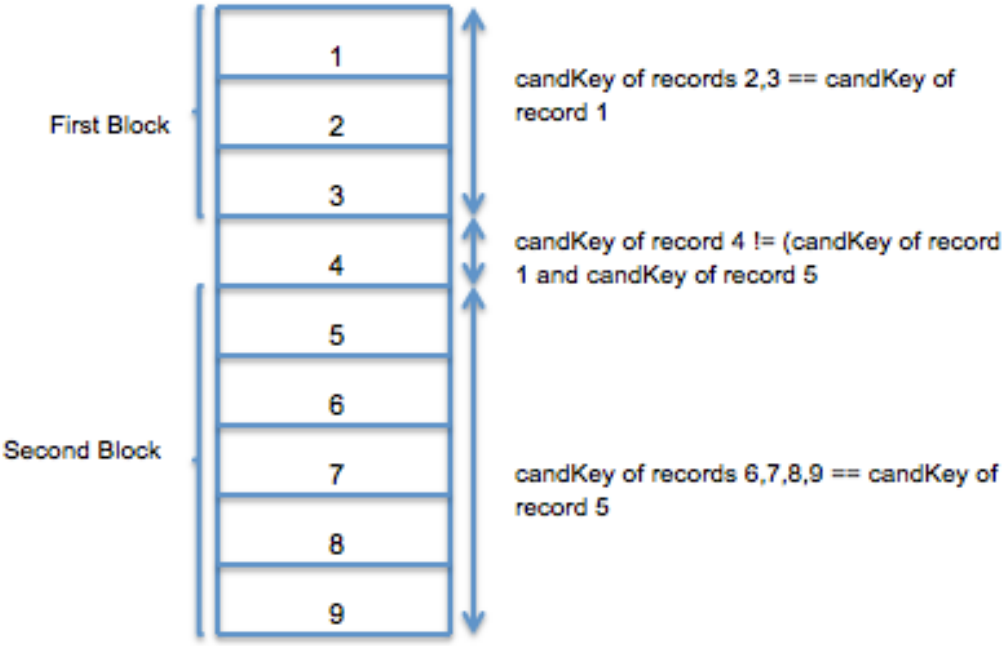

Figure 4.1 Creation of blocks using the blocking technique. candKey refers to the candidate key of a record. Image from [17].

The Sorted Neighborhood method is another technique used to reduce the number of comparisons and increase efficiency. The first step is to sort the records with the use of a sorting key, or candidate key. The key for each record is usually computed by extracting relevant attributes or portions of the attributes' values [17]. The idea behind the use of candidate keys is that common data will have a closely matching key. Once sorted, a fixed size window traverses the list of records. Each record will only be compared to other records within the window. For example, if the window size is w, each new record that enters the window will only be compared to $w-1$ records [10]. This is shown in Figure 4.2. After all possible comparisons within the window are complete, the window will slide

down by one, removing the first record and adding the next record in the list to the window. This continues until the window traverses over the entire dataset.

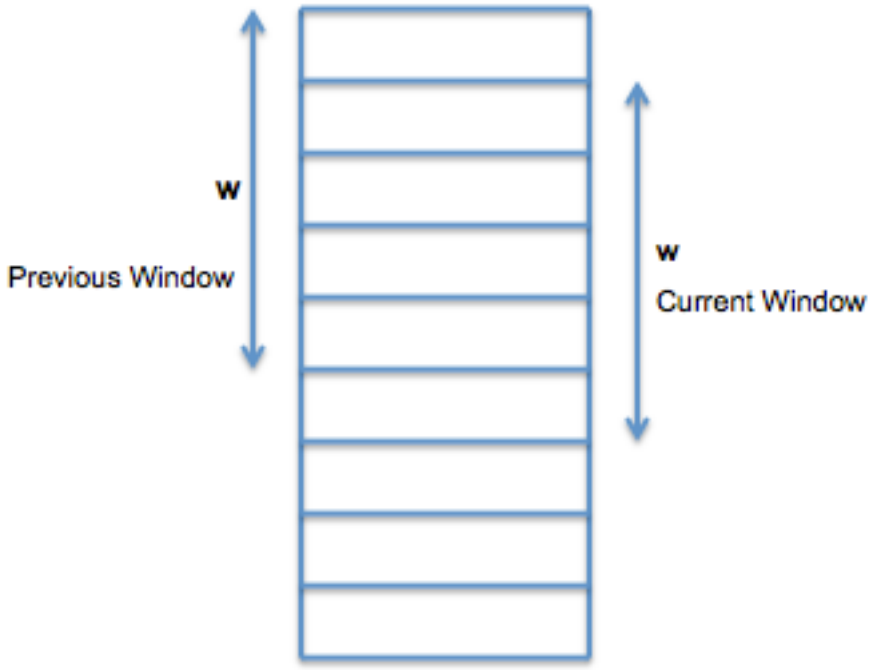

Figure 4.2 An example of the Sorted Neighborhood method. Image from [17].

4.2 String Similarity Metrics

Duplicate records are found by comparing attributes and determining if they are similar. This comparison is done through the use of string comparison metrics. Due to typographical and data entry errors, it is not feasible to compare attributes for equality, but rather to compare them for similarity. The Needleman and Wunsch metric is an edit distance metric that measures the amount of differences between two data strings [17]. It calculates the number of operations needed to transform one string to another and allows for varying costs for different edit distance operations.

CHAPTER 5

Cocktail Approach

This thesis is heavily based upon the work of Shirley Ong Ai Pei [17]. Pei presented an approach for general record linkage, called the Cocktail Approach. This approach combines techniques from a rule-based approach and a probabilistic-based approach. Statistics given by a probabilistic-based approach can give a good indication about the behavior of the dataset. Once a deep understanding of the data is obtained, a ruleset can be created to match records. The idea is to use statistics to gain a better understanding of the dataset and generate an equational theory ruleset. A training dataset may not mirror the behavior of another dataset exactly, but the dataset to be tested does reveal information about itself and its behavior. A probabilistic-based approach can also reveal which attributes make the most contributions to a mapping and which attributes do not [17]. The Cocktail Approach consists of two main phases: the probabilistic-based phase, and the rule-based phase.

In the first phase, the probabilistic-based phase, the first step is to sort the records with a candidate key, which is created from a combination of record attributes. Once all the records in the dataset are sorted, each record is placed into a block by computing a similarity score between the candidate keys of each record using the Needleman and Wunsch metric. The use of comparison subspaces is a way to reduce the number of candidate record pairs to a feasible number whilst maintaining the accuracy of the system [17]. All blocks are non-overlapping windows. For every block created, each record within the block is

compared with every other record in the block. A binary comparison vector, γ, is created for each candidate record pair in the block. Each attribute i in a record is compared to the equivalent attribute in another record using the Needleman and Wunsch metric. $\gamma^j{}_i = 1$ if attribute i agrees for record pair j, and $\gamma^j{}_i = 0$ if attribute i disagrees for record pair j. All vector patterns γ^j are stored with their respective frequency counts $f(\gamma^j)$. The frequency count $f(\gamma^j)$ represents the number of record pairs that generated the vector pattern γ^j.

The comparison vectors are then used by the EM algorithm to estimate values of m_i. Pei did not utilize the u_i value that was derived by the EM algorithm because it is derived from a biased situation. Instead, Pei used an alternative method to estimate the value of u_i by considering a random sample of record pairs in the dataset. u_i is estimated by randomly retrieving N number of records from the dataset where N is the number of record pairs used to estimate m_i. Comparison vectors are computed for N record pairs and is used to estimate the value of u_i. We can deduce that a low value of m_i is due to missing data, frequent mistakes of entering data into attribute i, or the information obtained for attribute i is simply not correct most of the time. If u_i is high, it is probably because the data is very common throughout the whole dataset for attribute i [17]. The values of m_i and u_i are used to calculate weights for each attribute, which in turn is used to determine which attributes are significant and insignificant. The attributes that are considered insignificant are not included in generating the equational theory ruleset, or the matching ruleset.

Leveraging the EM algorithm and making use of the vector pattern γ^j and their respective frequency counts $f(\gamma^j)$, a new matching ruleset can be generated. A vector pattern is considered to be a good matching rule if the frequency count of a matching vector pattern is significantly higher than the frequency count of a non-matching vector pattern. The ruleset generated by the probabilistic-based phase is then used by the second phase, the rule-based phase, to find duplicate records [17].

Once the ruleset has been generated, the second phase begins by using the Sorted Neighborhood method to traverse the sorted dataset for a second time. For each pair candidate created, another binary comparison vector is generated. This new comparison vector is compared against the matching ruleset. If the comparison vector matches a rule, then the records pair is classified as a match.

Pei tested the Cocktail Approach on three different datasets: Cora, Restaurant, and DBGen. The Cora dataset is a collection of citations to computer science papers. The Restaurant dataset is a collection of restaurant records. DBGen is a tool that generates datasets consisting of random US mailing addresses. For all three datasets, Pei was able to obtain an average recall score of about 96.8%. The recall score shown in (5.1) is the fraction of duplicates correctly classified over the total number of duplicates in the dataset [17]. Pei obtained an average precision score of about 97.9%. The precision score, shown in (5.2), is the fraction of correct duplicates over the total number of records pairs classified as duplicates. Finally, Pei obtained an average f-score of about

97.34%. The f-score, shown in (5.3) is the harmonic mean of the precision and recall values [17].

$$Recall = \frac{truePositives}{truePositives + falseNegatives} \quad (5.1)$$

$$Precision = \frac{truePositives}{truePositives + falsePositives} \quad (5.2)$$

$$F - Score = \frac{2 * Precision * Recall}{Precision + Recall} \quad (5.3)$$

This section gave a high level overview of the Cocktail Approach presented by Pei. Refer to [17] for a detailed explanation and implementation details.

CHAPTER 6

Implementation

My contributions include the adaption and evaluation of the Cocktail Approach in the genealogical domain. I chose the Cocktail Approach because it provided a way to generate matching rulesets without requiring training data. Record linkage in the genealogical domain is similar to record linkage in other domains in the sense that attributes or data associated with an entity are compared for similarities. However genealogical records behave differently because of similarities among different members of a society and family.

Along with the ideas from Pei, I incorporated the ideas from Wilson [21] into my implementation. Specifically, I modified the Cocktail Approach to not only analyze the data or attributes for each individual, but also analyze all relationships among relatives. There are common naming conventions among members of a society or in a family causing different records to be falsely classified as a match. Analyzing relationships will increase the confidence of matches and eliminate many false positives.

Finally, I created general domain metarules. The metarules include rules to automatically fill in missing data for the birth and death date fields if one was given and the other was unknown. If the approximate time period an individual has lived is known, then the birth and death date of the individual can estimated. By filling in the missing data, the system will be able to differentiate individuals from different time periods. In addition, when comparing two records, if

difference in birth and/or death is greater than the average lifespan, then the attributes will be considered non-matching.

Another metarule includes a giving priority to attributes that must never be considered to be insignificant by the system. As explained in the previous chapter, the EM algorithm computes the values m_i and u_i. These values are then used to determine which attributes are insignificant to the matching process and should be ignored. As a result, attributes such as first name, last name, and/or gender may be considered insignificant which will cause a problem. To evade this possibility, I created a priority list of attributes that must never be considered insignificant.

A detailed explanation of my implementation is given in the following sections and Figure 6.1 presents a diagram of the system architecture. Similar to the Cocktail Approach, my system is divided into two phases: the probabilistic-based phase and the rule-based phase. The first phase of my system is the probabilistic-based phase. Firstly, the dataset of genealogical records is sorted using candidate key, then divided into mutually exclusive blocks. Candidate record pairs are extracted from each block. For each candidate record pair, a binary comparison vector and frequency count is computed. The frequency count represents the number of record pairs that had generated the respective vector pattern. The comparison vectors and respective frequency counts are used by the EM algorithm to compute the probability of a match, m_i, and the probability of a non-match, u_i. m_i and u_i are then used to generate a list of insignificant attributes and a equational theory ruleset, also called a matching ruleset.

Once a matching ruleset is generated, the second phase of the system, the rule-based phase begins. The Sorted Neighborhood method is used to traverse the sorted list of records for a second time and create new candidate record pairs. A binary comparison vector is computed for each record pair and then compared against the matching ruleset. If the comparison vector matches a rule, then the record pair will be sent to the classifier and classified as a match. The following sections give a detailed explanation of the implementation of my system.

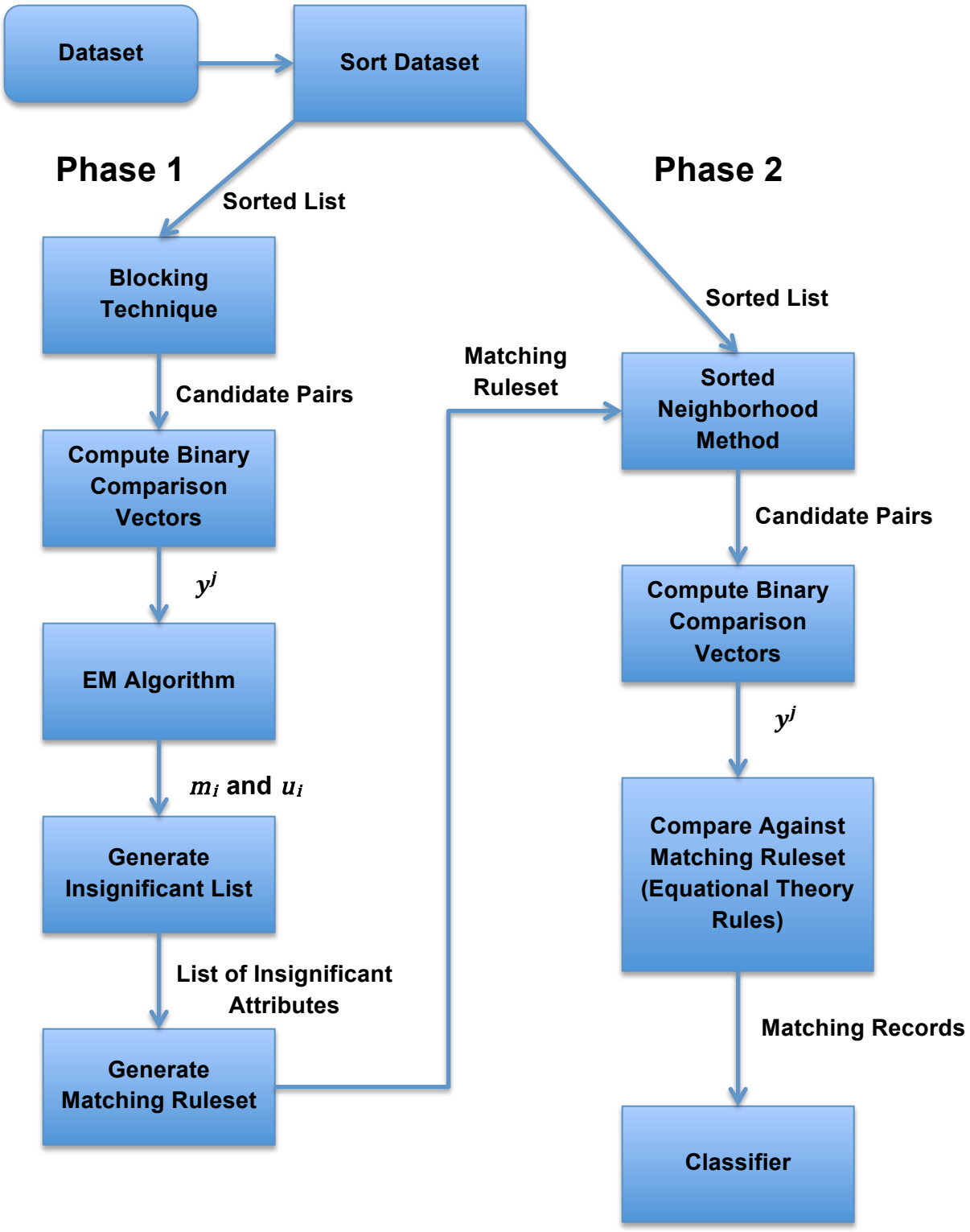

Figure 6.1 System Architecture [17].

6.1 Individual.py

The open-source python module used to parse GEDCOM files was created by Madeleine Price Ball and based on a parser written by Daniel Zappala at Brigham Young University. This parser was used to parse all GEDCOM files to extract all records and relationship data. Individual.py contains a class to represent a single individual and his/her family members and relatives.

```
self.first_name = ""
self.middle_name = ""
self.last_name = ""
self.birth = {'date': '' , 'month': '', 'year': '', 'place': ''}
self.death = {'date': '' , 'month': '', 'year': '', 'place': ''}
self.burial = ()
self.gender = ""
self.family = {}
self.candKey = ""
```

Figure 6.2 The representation of attributes for each genealogical record.

Figure 6.2 shows how the attributes of each genealogical record are represented. The *burial* attribute contains information about the burial date and place. Four types of family relationships are stored in the *family* dictionary; natural parents, parents, ancestors, and marriage information. Some records contain information about birth parents and adoptive parents. If both sets of parents are found, then the birth parents are be placed under the *Natural Parents* key in the dictionary and the adopted parents are be placed in the *Parents* key. If only one pair of parents is found, then they are placed under the *Parents* key. All ancestors found in an individual's family tree are placed under the *Ancestors* key,

and an individual's marriage information is placed under the *Marriage* key. The *candKey* is computed by combining the first three letters of an individual's first name with the first three letters of an individual's last name.

In many of the records, the birth date and/or death date fields are incomplete. For all records, if one of the dates is given, then the other date will automatically be filled in. The artificial_birth() function shown in Figure 6.3 fills in missing fields in the birth date attribute of the record if the birth date is unknown but the death date is known. The artificial_death() function fills in the missing fields in the death date attribute if the death date is unknown but the birth date is known.

```
def artificial_birth(self):

    for key in self.death.keys():
        if key == 'year':
            if self.birth['year'] == -1 and self.death['year'] != -1:
                self.birth['year'] = self.death['year'] - avg_lifespan(self.death['year'])
        elif len(self.birth[key]) == 0 and len(self.death[key]) != 0:
            self.birth[key] = self.death[key]


def artificial_death(self):

    for key in self.birth.keys():
        if key == 'year':
            if self.death['year'] == -1 and self.birth['year'] != -1:
                self.death['year'] = self.birth['year'] + avg_lifespan(self.birth['year'])
        elif len(self.death[key]) == 0 and len(self.birth[key]) != 0:
            self.death[key] = self.birth[key]
```

Figure 6.3 Functions to automatically fill in missing information about an individual's birth or death date.

6.2 needleman_wunsch.py

The NeedlemanWunsch class represents the string similarity metric used for all string similarity calculations. It takes in a pair of strings and returns the distance result in the form of a floating point number between 0.0 and 1.0. The methods

are based off of the Needleman-Wunsch distance metric in the SimMetrics java module, an open source Java library of distance metrics.

6.3 comparison_space.py

The Compare_space class is responsible for sorting the dataset and creating candidate record pairs for comparison. The records are sorted using a *candidate key (candKey)*. This class also keeps track of all comparison vector patterns created, the ruleset generated, and list of insignificant attributes. Figure 6.4 shows how each record is classified in a block. Each record in a block will only be compared to other records within the same block. For each candidate pair created, a binary comparison vector is computed and used in the first phase of the Cocktail Approach. Figure 6.5 shows how records are clustered together using the Sorted Neighborhood method. Each record is compared to other records within the same window and all pairs are used in the second phase of the Cocktail Approach. Both functions are based on the implementation by Pei [17].

```
def create_blocks(self):
    '''
        ALGORITHM 1

        Input: all sorted records from dataset
        Output - Identifiers of all records in a block

        Note: all blocks are non-overlapping windows
    '''

    blocks_list = []
    nw = NeedlemanWunsch()
    threshold = 1
    i = 0

    while i < len(self.sorted_list):

        record_A = self.sorted_list[i].getCandKey()
        blockSize = 1

        for j in range(i+1, len(self.sorted_list)):
            record_B = self.sorted_list[j].getCandKey()
            sscore = nw.getSimilarity(record_A, record_B)
            if sscore >= threshold:
                blockSize += 1
            else:
                break                                    #quit inner loop

        if blockSize > 1:

            ids = self.sorted_list[i:(i+blockSize)]          #ids <- all ids in block
            blocks_list.append(ids)
        else:
            pass

        i += blockSize

    return blocks_list
```

Figure 6.4 Function to divide dataset of records into non-overlapping blocks.

```
def slidingWindow(self, win_size):
    '''
        ALGORITHM 2 - Sliding Window
        Input: sliding window size (winSize)

        Create all candidate pairs and forward each to phase 2 of Cocktail
    '''

    dbSize = len(self.sorted_list)
    totalSlides = dbSize - win_size

    for i in range(0, totalSlides):
        ids = self.sorted_list[i:i+win_size]
        self.pairRecords_slide(ids)
```

Figure 6.5 Function that clusters records together using the Sorted Neighborhood method.

6.4 cocktail.py

Similar to the cocktail approach presented by Pei, my core system consists of two phases.

6.4.1 Phase 1

In the first phase, binary comparison vectors for each candidate record pair are generated and utilized to estimate values m_i and u_i. Based on these values, a list of insignificant attributes and a matching ruleset are generated. A set of randomly selected candidate record pairs is used to estimate an alternative value for u_i. The number of random candidate pairs is equal to the number of candidate pairs used to estimate the value of m_i. Figure 6.6 shows how an alternative value for u_i is computed.

```
def computeUI(vector_patterns, totalPairs, dataList):
    '''
        ALGORITHM 8
        vector_patterns = A collection of vector patterns and their respective frequency counts
        output u_i
    '''

    u_i = [0.1] * TOTAL_ATTRIBUTES

    c = {}

    for x in range(0, totalPairs):
        t1 = dataList[random.randint(0, len(dataList)-1)]
        t2 = dataList[random.randint(0, len(dataList)-1)]
        while t1 != t2:
            t2 = dataList[random.randint(0, len(dataList)-1)]

        y_j = repr(compute_vector_pattern(t1, t2))

        if y_j in c.keys():
            #add 1 to the frequency counts for y_j
            c[y_j] += 1
        else:
            #insert y_j and its frequency count to 1 to c
            c[y_j] = 1

    for i in range(TOTAL_ATTRIBUTES):
        counter = 0
        for k in vector_patterns.keys():
            y_j = eval(k)               #string representation to list representation

            if y_j[i] == 0:
                counter += vector_patterns[k]

        u_i[i] = counter/totalPairs
```

Figure 6.6 Function to compute an alternative value of u_i.

The values of m_i and u_i are used to calculate weights of each attribute among all comparison vectors. In other words, for each comparison vector, if attribute i matches (i.e. is 1), then the weight is calculated by:

$$w_i = log_2\left(\frac{m_i}{u_i}\right) \tag{6.1}$$

If attribute i does not match (i.e. is 0), then the weight is calculated by:

$$w_i = log_2\left(\frac{1-m_i}{1-u_i}\right) \tag{6.2}$$

The total weights of each attribute are then compared to determine which attributes are insignificant to the matching process. An attribute is considered insignificant if it has a very low weight score compared to the rest of the attributes

[17]. Pei claims that a very low weight score will not give any contributions to the matching rules.

The last step in the first phase is to generate a matching ruleset. This is done by observing the vector patterns, γ^j, and their respective frequency counts, $f(\gamma^j)$. Figure 6.7 shows how a ruleset is generated. A vector pattern makes a good matching rule if the frequency count for a matching record pair is sufficiently higher than the frequency count of a non-matching record pair. This is determined by taking the logarithm of the relative frequency over the logarithm of the total number of vector patterns. The results are then compared against two decision rules: *t1* and *t2* which are pre-defined. If the vector patterns measure up to the relative weight of at least *t1* and relative error of at most *t2*, then it will be added to the matching ruleset.

```
for k in mCollectionVecPattern.keys():

    y_j = eval(k)               #string representation to list representation

    #Declaring values
    mFrequency = uFrequency = 0.0
    minU = 0.0

    mFrequency = mCollectionVecPattern[k]

    if k in uCollectionVecPattern.keys():
        uFrequency  = uCollectionVecPattern[k]
    else:
        uFrequency = 0.0

    if uFrequency == 0:
        minU = 0.0
    else:
        minU = uFrequency/totalPairs

    #can't take log of zero or log of a negative zero
    if (mFrequency - uFrequency) <= 0:
        continue

    if float(math.log(mFrequency-uFrequency)/math.log(totalPairs)) >= t1 and minU <= t2:

        #modify y_j to change from 1 to 0 for insignificant attributes based on insignificantList
        for insig in insignificantList:
            y_j[insig] = 0

        #count the total number of attributes with 1 and store y_j together with the total count in list X
        total_1 = 0
        for i in range(TOTAL_ATTRIBUTES):
            if y_j[i] == 1:
                total_1 += 1
        temp = (y_j, total_1)
        X.append(temp)

        if y_j not in Z:
            Z.append(y_j)


#return a ruleset Z consisting of y_j as matching rules
return Z
```

Figure 6.7 Generates a matching ruleset using comparison vector pairs and their respective frequency counts.

6.4.2 Phase 2

The second phase of my system uses the Sorted Neighborhood method to create new candidate record pairs. A binary comparison vector is computed for each candidate record pair and compared against the matching ruleset to determine whether the pair matches or not. This comparison consists of checking if every attribute that is a match in a rule is also a match in the pair's comparison vector. The comparison vector is compared against all rules in the ruleset until a

match is found. The idea is that the rules signify the minimum matches that a pair should have in order to be classified as a match. Figure 6.8 shows how each comparison vector is compared to the matching ruleset. Figure 6.9 shows how the check is done. All record pairs that are considered a match are forwarded to the classifier, which outputs the records pairs. All record pairs that contain a gender mismatch are indicated in the output.

```
def compare_rules(t1, t2, rule_set, insignificantList):
    '''
        ALGORITHM 11 - For Rule Based Part
        Input record pair t1, t2
        Output matching record pair t1, t2
    '''


    #Vector pattern indicates which attributes are similaiar and which aren't
    y_j = compute_vector_pattern(t1, t2)

    #modify y_j to change from 1 to 0 for insignificant attributes based on insignificantList
    for insig in insignificantList:
        y_j[insig] = 0

    #pass similarity scores to Equational Theory rules of dataset
    #if (t1, t2) is found to be matching:
    if check_rules(y_j, rule_set):
        classifier(t1,t2)
```

Figure 6.8 Compares a comparison vector representing a record pair against a matching ruleset.

```
def check_rules(y_j, rule_set):

    for rule in rule_set:
        res = True
        for i in range(0, TOTAL_ATTRIBUTES):
            if rule[i] == 1:
                if rule[i] != y_j[i]:
                    res = False

        if res:
            return True

    return False
```

Figure 6.9 Function to compare a comparison vector against all rules within a ruleset.

CHAPTER 7

Testing and Evaluation

The system was tested on multiple artificially created datasets consisting of GEDCOM files. I was only able to obtain a single dataset of GEDCOM files because genealogical records are not commonly published or shared with the public. This dataset of GEDCOM files used to test and evaluate my system was provided by Western Michigan Genealogical Society from its collection and made available by Roger Moffat of WMGS. Due to privacy concerns, the dataset will not be published. The artificial datasets created for testing contains a random subset of GEDCOM files from the dataset obtained. More information about each dataset is given in the nest section. The recall, precision, and f-score metrics defined in [17] and in Chapter 6 are used to evaluate the performance of the Cocktail Approach on genealogical records.

7.1 Datasets

All testing was done using GEDCOM files from a single dataset. The dataset contains a total of 255,181 records. All files consist of individuals between the 1500s and 1990s. This dataset accurately mirrors real-world issues encountered in GEDCOM files such as missing fields, typographical errors, or data entry errors. Testing my implementation on this dataset will give insight on how well the Cocktail Approach performs on genealogical data. Due to performance issues all testing was done on a subset of the records. Refer to Section 7.4 for more details on performance. Information on each dataset is presented below:

- Dataset 1 contains 5 unique GEDCOM files and a total of 10 files. Each file has been duplicated and modified to artificially create duplicate records that are similar but not identical. A total of 190 records are contained in the dataset.
- Dataset 2 contains 17 unique files and a total of 34 files. Each unique file was duplicated to create identical duplicates. A total of 696 records are contained in the dataset.
- Dataset 3 contains 76 unique files and a total of 152 files. Each unique file was duplicated to create identical duplicates. A total of 16,816 records are contained in the dataset.
- Dataset 4 contains 18 total files and a total of 945 records. Six files were duplicated and modified resulting in 351 duplicate records.
- Dataset 5 contains 75 total files and a total of 1,625 records. 20 files were duplicated and modified resulting in 320 duplicate records.
- Dataset 6 contains 133 total files and a total of 3,406 records. 45 files were duplicated and modified resulting in 1,118 duplicate records.
- Dataset 7 contains 20 total files and a total of 359 records. Four files were duplicated and modified resulting in 76 duplicate records.
- Dataset 8 contains 63 total files and a total of 7,689 records. Eleven files were duplicated and modified resulting in 320 duplicate records.
- Dataset 9 contains 59 total records and a total of 1,794 records. Seven files were duplicated and modified resulting in 186 duplicate records.

- Dataset 10 contains 30 duplicate records and a total of 1,108 records. One file was duplicated and modified resulting in 222 duplicate records.
- Dataset 11 contains 12 total files and a total of 594 records. No files were duplicated.
- Dataset 12 contains 55 total files and a total of 1,305 records. No files were duplicated.
- Dataset 13 contains 88 total files and a total of 2,288 records. No files were duplicated.

Table 7.1 shows the number of missing attributes in each dataset. For example, Dataset 1 contains 190 records and 96 of the records is missing the middle name attribute. The number of missing attributes shown in Table 7.1 are the same for before and after the addition of metarules to to the system. Table 7.2 shows the number of missing birth date and death date attributes for each dataset for the baseline system. Table 7.3 shows the number of missing attributes for each dataset after the addition of metarules. Recall that the metarules attempts to fill in the missing information if one of the dates is given. By filling in the missing information, more data will be used during the comparision, increasing the confidence of matches.

Dataset	Num Records	First Name	Middle Name	Last Name	Gender	Burial	Family Natural Parents	Family Parents	Family Ancestors	Family Marriage
1	190	2	96	10	0	190	190	100	100	124
2	696	2	358	26	2	696	696	350	350	302
3	16816	156	10050	496	2	16614	16816	6598	6598	8794
4	945	1	589	77	4	945	945	467	467	453
5	1625	3	781	49	0	1613	1625	865	865	484
6	3406	8	1721	114	0	3394	3406	1768	1768	1165
7	359	0	159	5	0	359	359	198	198	94
8	7689	78	4597	188	1	7590	7689	2967	2967	4012
9	1794	6	964	71	0	1789	1794	923	923	656
10	1108	5	700	68	0	1095	1108	553	553	418
11	594	1	369	48	0	594	594	296	296	280
12	1305	3	661	44	0	1293	1305	686	686	403
13	2288	7	1171	83	0	2272	2288	1189	1189	782

Table 7.1 The number of missing attributes in each dataset. The attributes in this table are not affected by the addition of the metarules.

Dataset	Num Records	Birth Date	Birth Month	Birth Year	Birth Place	Death Date	Death Month	Death Year	Death Place
1	190	80	80	68	68	104	104	94	100
2	696	204	204	164	310	314	314	288	412
3	16816	7122	7122	5482	8390	11978	11978	11208	11788
4	945	444	444	295	256	486	486	403	457
5	1625	436	436	341	429	670	670	565	656
6	3406	1087	1087	828	1115	1517	1517	1303	1602
7	359	84	84	70	90	125	125	122	141
8	7689	3241	3241	2504	3936	5572	5572	5252	5972
9	1794	720	720	542	803	905	905	768	1066
10	1108	451	451	35	411	523	523	439	541
11	594	275	275	189	164	302	302	251	251
12	1305	377	377	291	374	543	543	457	542
13	2288	760	760	574	765	1037	1037	887	1084

Table 7.2 The number of missing attributes for each dataset before the addition of metarules. The birth and death date attributes are affected by the general domain metarules.

Dataset	Num Records	Birth Date	Birth Month	Birth Year	Birth Place	Death Date	Death Month	Death Year
1	190	68	68	56	54	68	68	56
2	696	160	160	134	272	160	160	134
3	16816	6466	6466	5136	7934	6466	6466	5136
4	945	357	357	258	217	357	357	258
5	1625	330	330	278	307	330	330	278
6	3406	886	886	705	909	886	886	705
7	359	59	59	59	58	59	59	59
8	7689	2942	2942	2370	3777	2942	2942	2370
9	1794	602	602	464	731	602	602	464
10	1108	338	338	272	321	338	338	272
11	594	222	222	163	139	222	222	163
12	1305	282	282	233	278	282	282	233
13	2288	618	618	488	626	618	618	488

Table 7.3 The number of missing attributes for each dataset after the addition of metarules. Birth and death fields are affected by the metarules.

As explained in Chapter 1, genealogical records tend to contain several inconsistencies among the data such as typographical errors, data entry errors, missing data, and/or lexical heterogeneity. For testing purposes, I artificially created duplicates within 10 of the datasets and modified the data to mirror these inconsistencies. The modifications to the records include deleting fields to artificially create missing data, and modifying original data to insert typographical and data entry errors. Modifications were made to the data to create duplicate records that are similar, but not identical.

Typographical and data entry errors were inserted into the data by changing one or two letters in the string data for a few attributes in the duplicate records. For example, if an individual's first name is *Natalie*, the first name attribute in the duplicated record was changed to *Natlie* or *Natalia*. Data entry

errors were inserted by inserting or deleting one or two letters in the string data. Typographical errors were inserted by modifying one or two letters. For a random subset of duplicated record in each dataset, a few attributes, including the birth and death dates, were randomly chosen and modified.

Matching duplicated records that are poorly document is a great challenge for genealogical record linkage. Many records, especially records of women, are poorly documented and have missing information. For a random subset of duplicate records, if the records do not already have missing data, one or two fields were deleted. These modifications were created to mirror real world issues found within genealogical records and evaluate how well the Cocktail Approach performs on poorly documented data.

7.2 Baseline Results

Dataset	Num Files	Num Records	Num Duplicate Records	Matches Found	True Positives	False Positives	False Negatives	Gender Mismatch
1	10	190	95	0	0	0	95	0
2	34	696	348	152	75	77	273	24
3	152	16,816	8,408	14514	8400	6114	8	2300
4	18	945	351	783	323	460	28	160
5	75	1,625	320	1045	302	743	18	213
6	133	3,406	1,118	2,884	1118	1766	0	520
7	20	359	76	187	71	116	5	77
8	63	7,689	320	11,702	320	11382	0	3670
9	59	1,794	186	1428	185	1243	1	364
10	30	1,108	222	894	208	686	14	212
11	12	594	15	44	15	29	0	2
12	55	1,305	19	993	19	974	0	310
13	88	2,288	7	1,777	7	1770	0	509

Table 7.4 Baseline testing results of each dataset.

Dataset	1	2	3	4	5	6	7	8	9	10	11	12	13
F-Score	0%	29.9%	73.2%	56.9%	44.2%	55.8%	53.9%	5.25%	22.8%	37.2%	50.8%	3.7%	0.8%
Recall	0%	21.5%	99.9%	92%	94.3%	100%	93.4%	100%	99.4%	93.7%	100%	100%	100%
Precision	0%	49.3%	57.8%	41.2%	28.9%	38.7%	37.9%	2.7%	12.9%	23.2%	34.1%	1.9%	0.4%
Percent Duplicates	50%	50%	50%	37%	19%	32%	21%	4%	10%	20%	2.5%	1.4%	0.3%

Table 7.5 Baseline recall scores, precision scores, f-scores, and percent of duplicates for each dataset.

The Cocktail Approach was first tested on each dataset without incorporating any modifications. In other words, the system did not compare relationships of any individuals or incorporate general domain metarules. The results from the first

test run are shown in Table 7.4 and Table 7.5 shows the precision scores, recall scores, f-scores, and the percent of duplicates for each dataset. These results are the baseline results of the Cocktail Approach in the genealogical domain. These baseline results will be used to evaluate the reimplementation and adoption of the Cocktail Approach to genealogical records. The average recall score obtained by the baseline system is 85.1% and the average precision score is 25.3%. The average f-score of the baseline system is 33.4%. The Cocktail Approach obtained a high number of false positives resulting in very low precision scores. The low precision score is directly related to the percentage of true duplicate records within the dataset. A detailed explanation of this relationship is given in Section 7.3. The recall scores obtained by the baseline system are adequate for majority of the datasets. However, there are a few exceptions. Dataset 1 did not classify any record pairs as duplicates and Dataset 2 only found 75 of the true duplicates in the dataset.

7.3 Results

Dataset	Num Files	Num Records	Num Duplicate Records	Matches Found	True Positives	False Positives	False Negatives	Gender Mismatch
1	10	190	95	95	89	6	6	0
2	34	696	348	257	248	9	100	0
3	152	16,816	8,408	14,514	8,408	6,106	0	2,300
4	18	945	351	382	348	34	3	0
5	75	1,625	320	814	320	494	0	0
6	133	3,406	1,118	2,884	1,085	1799	33	19
7	20	359	76	68	58	10	18	0
8	63	7,689	320	11,702	302	11,400	18	3,670
9	59	1,794	186	798	186	612	0	5
10	30	1,108	222	521	222	299	0	0
11	12	594	15	44	15	29	0	0
12	55	1,305	19	495	19	476	2	0
13	88	2,288	7	1,777	7	1,770	0	0

Table 7.6 Testing results of each dataset.

Dataset	1	2	3	4	5	6	7	8	9	10	11	12	13
F-Score	93%	81%	72.6%	94.8%	56.1%	53.5%	80.2%	4.87%	37.4%	59.1%	50.7%	7.32%	0.7%
Recall	93%	71%	100%	99%	100%	97%	76%	94%	100%	100%	100%	100%	100%
Precision	93%	96%	57%	91%	39%	37%	85%	2.5%	23%	42%	34%	3.8%	0.4%
Percent Duplicates	50%	50%	50%	37%	19%	32%	21%	4%	10%	20%	2.5%	1.4%	0.3%

Table 7.7 Recall scores, precision scores, and percent of duplicates for each dataset.

Table 7.6 presents testing results of each dataset for my system. Table 7.7 shows the recall scores, precision scores, f-scores, and the percent of duplicates for each dataset. The recall score is the fraction of duplicate records correctly classified over the total number of duplicates in the dataset. The precision score is the fraction of correct duplicate records over the total number

of record pairs classified as a match. The f-score is the harmonic mean of the precision and recall scores [17]. My implementation of the Cocktail Approach received an average recall score of 94.6%, an average precision score of 46.4%, and an average f-score of 53.2%.

The average precision score is significantly lower than the average recall score and is due to the percent of true duplicates within a dataset. The matching ruleset is generated using the binary comparison vectors for all candidate record pairs in the dataset. A dataset with a low percentage of duplicates will not provide a good model for a matching rule. Without a sufficient amount of true duplicate records, the system will not be able to compute a comparison vector that represents a true duplicate. The majority of comparison vectors computed from a dataset with a low percentage of duplicates will only represent record pairs that may only have one or a few similar attributes. These vectors were added to the ruleset causing the system to classify all record pairs that only have a few similar attributes as a match, resulting in a high number of false positives and a low precision score.

The relationship between the precision score and the percent duplicate is shown when comparing the results of Dataset 4 and 8. Dataset 4 contains 37% of duplicates and received a precision score of 91%. Dataset 8 contains 4% of duplicates and received a precision score of 2.5%. Due to the low number of true duplicates, the system was not able to generate a good matching rule for Dataset 8. As a result, 97% of records that were classified as a match for Dataset 8 were wrong. The difference in the percentage of duplicate records between Dataset 4

and 8 was 33%. However, the matching ruleset generated for Dataset 4 was able to model true duplicate records and received a 91% precision score. Figure 7.1 shows a comparison of the percent of duplicates, precision scores, recall scores, and f-scores for each dataset. Datasets 9, 10, 11, 12, and 13 also received precision scores significantly lower than recall scores due to not containing sufficient data of duplicate records.

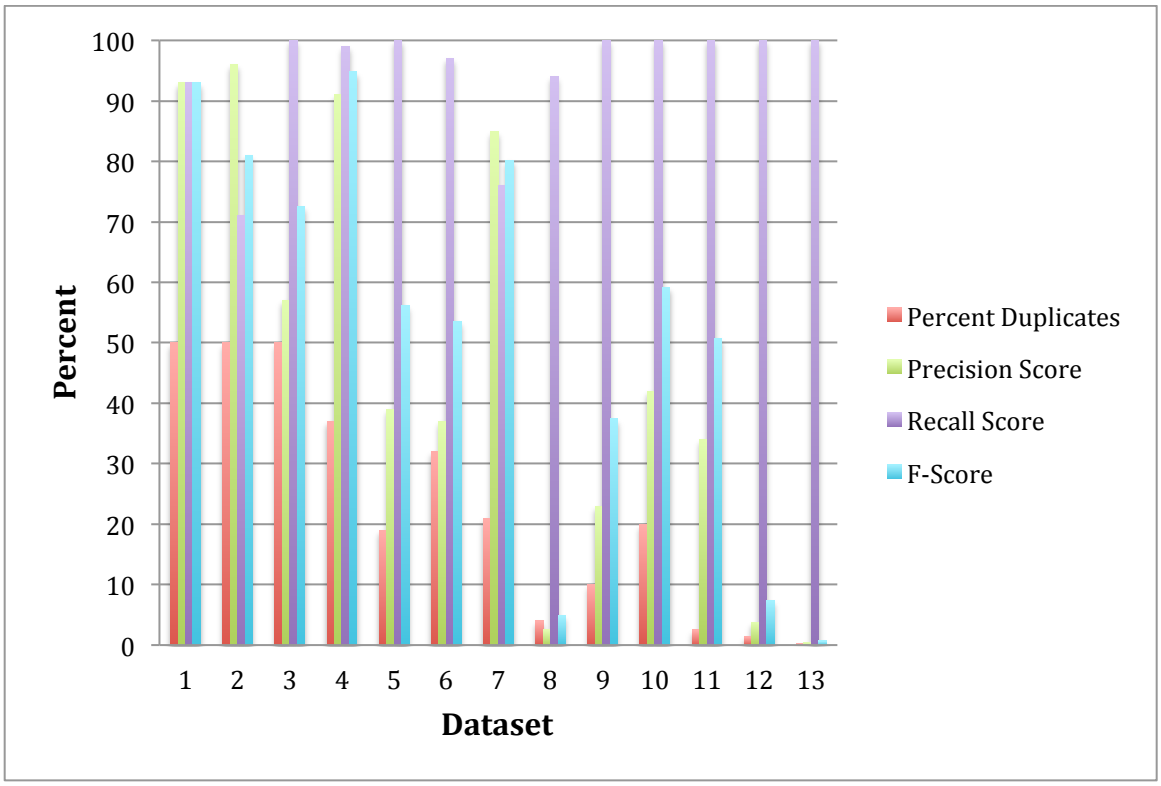

Figure 7.1 Comparison of precision scores, recall scores, and f-scores to the percent of duplicates for each dataset.

The recall scores for all datasets were relatively high. Datasets 9, 10, 11, 12, and 13 all received recall scores of 100%. This is because of the matching ruleset generated only searched for a few similarities between records. Therefore, all true duplicate records were correctly classified as a match.

Many of the false positives were records that had many missing attributes. Figure 7.2 shows two records that were wrongly classified as a match. These two individuals have the same first name but no other similar attributes. The first record only has information about the individual's birth and death date, and the second record does not have any other information about the individual. The combination of records with missing data and a bad ruleset dramatically increases the number of false positives.

```
Matching Pair:
Elizabeth
Elizabeth



First_Name Elizabeth
Middle_Name
Last_Name
Gender F
Birth_Date {'date': '', 'year': 1590, 'place': 'Los Angeles., California, USA', 'month': ''}
Death_Date {'date': '16', 'year': 1660, 'place': 'Los Angeles, California, USA', 'month': 'JUL'}
Burial ('', '', ())
Family_Dict {'Nat_Parents': [], 'Parents': [], 'Ancestors': [], 'Marriage': []}
Cand_Key Eli


First_Name Elizabeth
Middle_Name
Last_Name
Gender F
Birth_Date {'date': '', 'year': -1, 'place': '', 'month': ''}
Death_Date {'date': '', 'year': -1, 'place': '', 'month': ''}
Burial ('', '', ())
Family_Dict {'Nat_Parents': [], 'Parents': [], 'Ancestors': [], 'Marriage': []}
Cand_Key Eli
```

Figure 7.2 An example of a record pair that was falsely classified as a match.

Another common cause of false positives is due to common names among family members. Many families name children after relatives or parents. Figure 7.3 shows another record pair that was falsely classified as match. These two record pairs are of a parent and child. Majority of the time, a child and parent are born in or live in the same place causing those attributes to match. In addition, they have the same ancestors therefore the *ANCESTOR* attribute will

be classified as a match. Before searching for duplicate records, the system sorts the dataset by the *candKey*, which is a combination of parts of the first and last name for each individual. A majority of family members have the same last name, and similar first names causing the records to classified in the same block or window. When being compared, the system will find many similarities among the data and classify them as a match. Records of family members make up a significant number of the false positives found in all datasets.

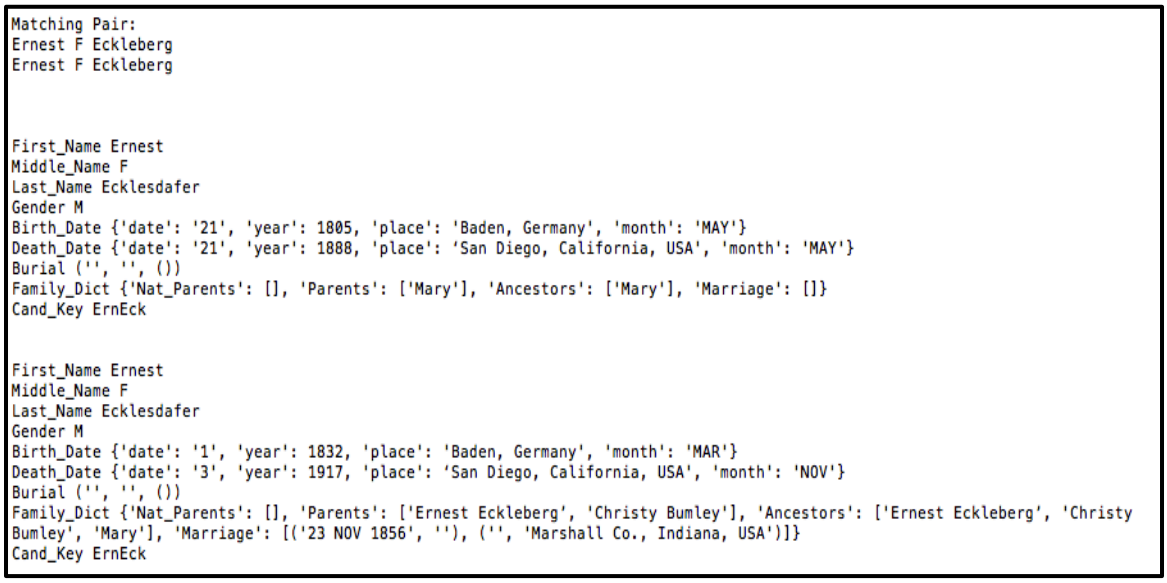

Figure 7.3 An example of records that are in the same family tree and were falsely classified as a match.

Although the average precision score was low, the average recall score is satisfactory. For every dataset, majority of the true duplicate records were found. Figure 7.4 shows an example of a true positive record pair. Regardless of the percent of duplicates in a dataset, the Cocktail Approach was able to find a majority of the true duplicate record pairs.

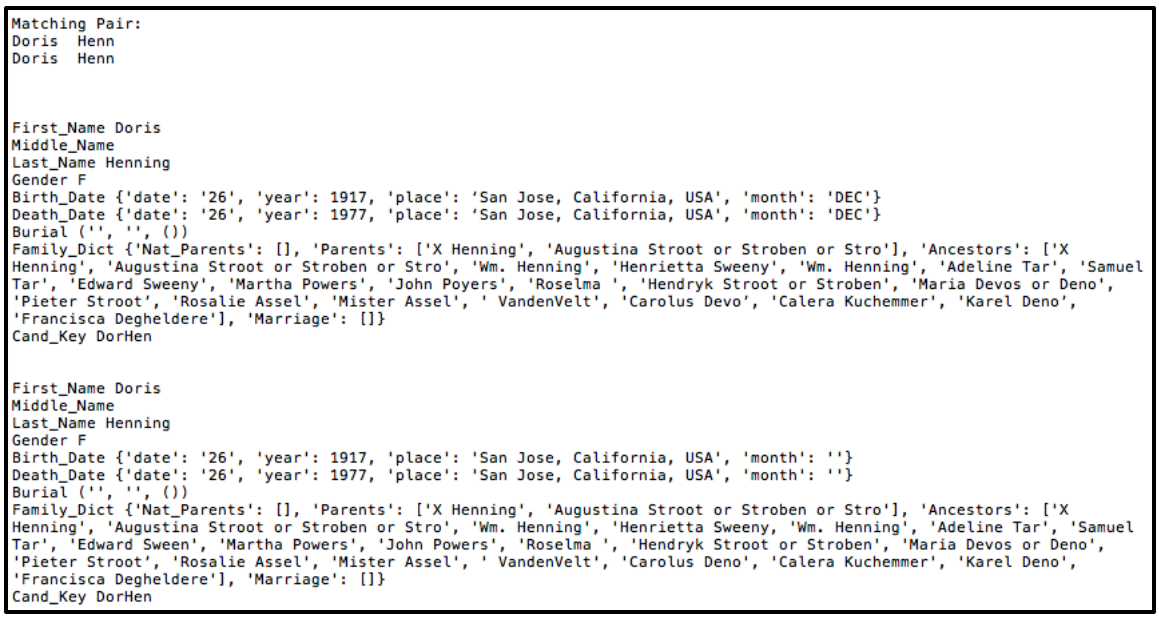

Figure 7.4 An example of a true positive record pair.

7.4 Insignificant Attributes

For each dataset, the attributes that were determined to be insignificant by the baseline system are shown in Table 7.8. The attributes determined to be insignificant by my system are shown in Table 7.9. The index-attribute mapping is shown in Figure 7.5.

```
FIRST_NAME  = 0
MIDDLE_NAME = 1
LAST_NAME   = 2
BIRTH_DATE  = 3
DEATH_DATE  = 4
BURIAL      = 5
GENDER      = 6
ANCESTOR    = 7
NAT_PARENTS = 8
PARENTS     = 9
MARRIAGE    = 10
```

Figure 7.5 Index-attribute mapping.

For all datasets, the NAT_PARENTS (natural parents) attribute was classified as insignificant. This is due to a majority of individuals only having one pair of parents, which was represented by the *PARENT* field. Therefore, the *NAT_PARENTS* field was left blank resulting in a very low m_i value and a very high u_i value compared to rest of the attributes. Majority of the datasets found the *BURIAL* attribute as insignificant also due to missing data. The *DEATH_DATE* and *MARRIAGE* attributes were sometimes also classified as insignificant due to missing data.

Without a metarule giving priority to certain attributes, the system may ignore important attributes during the matching process. The baseline system did not use any metarule and as a result, important attributes such as an individual's first name and last name were considered insignificant by the system. In addition, several attributes were completely ignored, reducing the amount of data to be compared during the matching process. This decreases the accuracy of the system and increases the number of false positives. My system used a priority list of attributes so important attributes were never considered insignificant. This decreased the number of attributes ignored during the matching process and the number of false positives.

Dataset	1	2	3	4	5	6	7	8	9	10	11	12	13
Insignificant attributes	0, 2, 5, 6, 8	0, 2, 5, 6, 8	8	0, 2, 5, 6, 8	0, 5, 6, 8	0, 2, 6, 8	0, 2, 5, 6, 8	8	0, 2, 6, 8	0, 2, 5, 6, 8	4, 5, 6, 8	2, 5, 6, 8	5, 6, 8, 10

Table 7.8 Attributes that were considered insignificant for each dataset for the baseline system.

Dataset	1	2	3	4	5	6	7	8	9	10	11	12	13
Insignificant attributes	5,8	5,8	8	5,8	5,8	8	8	5,8	8	5,8	4,5,8	5,8	5,8,10

Table 7.9 Attributes that were considered insignificant for each dataset.

7.5 Performance

The use of comparison subspaces is used to increase efficiency. Instead of comparing every record in the dataset to every other record, each record is only compared to a subset of records. Because the *candKey* of each record is a combination of parts of an individual's first and last name, all individuals with similar names will be placed in the same block. In most cases, these are individuals within a family tree. Figure 7.6 shows the performance time in seconds for each dataset. Dataset 1 contains 190 records and finished running in less than three seconds. Dataset 10 contains 1,108 records and finished running in about 35 seconds. However, Dataset 3 contains 16,816 records and took about 89 minutes to run. Once cause is because of common naming conventions among family members causing most records within a single family tree to be placed in the same block. The time complexity for comparisons for each block is $O(n^2)$. Increasing the number of family trees in the dataset increases the computation time exponentially. In addition, the Cocktail Approach traverses over the dataset creating candidate record pairs twice; once with the blocking technique and once with the Sorted Neighborhood method. Running my system on the entire dataset of 255,181 records was not feasible; hence I created 13 artificial datasets, each containing a subset of records from the original dataset to test and evaluate my system.

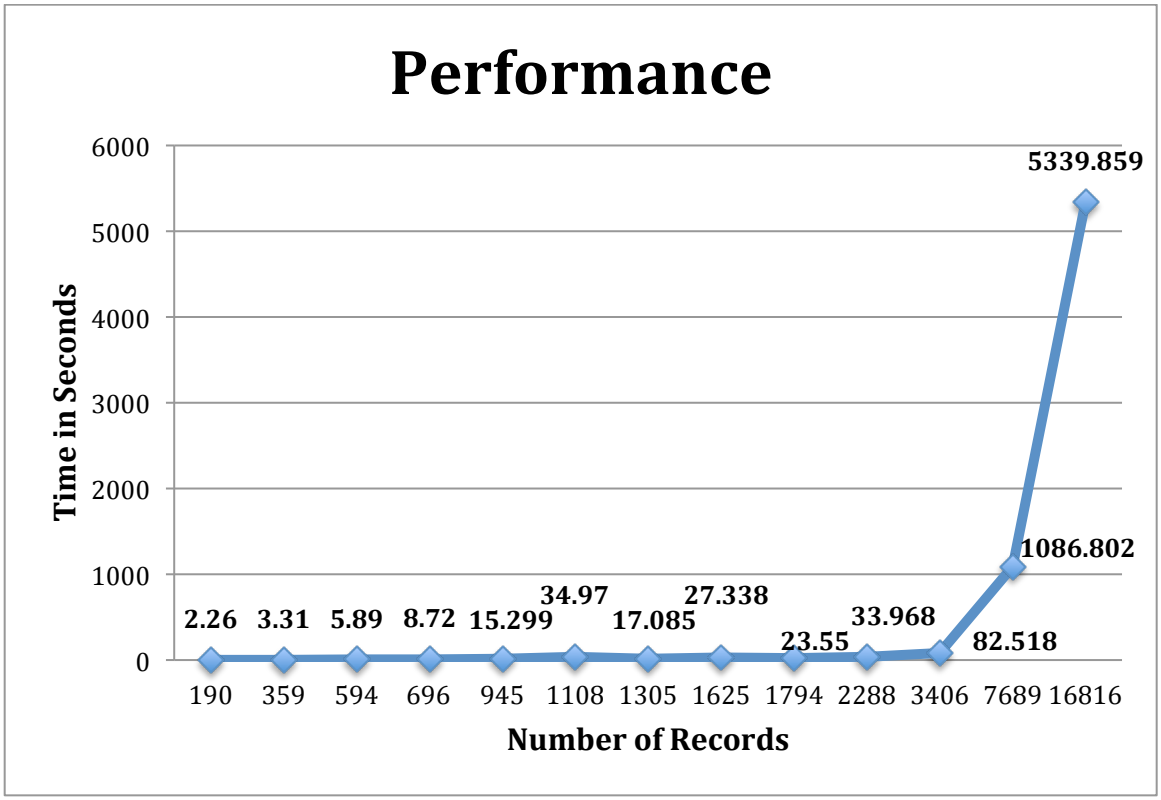

Figure 7.6 Performance time for each dataset.

7.6 Evaluation

The overall goal of this thesis was to find an efficient record linkage algorithm that reduces the need of human intervention and has a high accuracy of finding duplicate genealogical records within family trees. The baseline system received relatively low recall and precision scores. The average baseline recall score is 85.1%, the average baseline precision score is 25.3%, and the average baseline f-score is 33.4%. Although the system found majority of the true duplicate records, it still missed many true duplicate records. My implementation of the Cocktail Approach is also able to find majority of the duplicate records even with the modifications to the duplicate records in each dataset. Incorporating

metarules and analyzing relationships improves the performance so the Cocktail Approach for genealogical records. My system received an average recall score 9.5% higher than the average baseline recall score and an average precision score 21.1% higher than the average baseline precision score.

As mentioned before, general record linkage algorithms are not always applicable to different domains than which it was developed for. The Cocktail Approach provides a way to analyze the dataset and generate a matching ruleset without the need of manual work. However, this approach does not perform well in the genealogical domain. My modified version of the Cocktail Approach does provide a good way to analyze the dataset and find duplicates without the need of training data and account for common issues found within genealogical records. Majority of the true duplicate records were found thus significantly reducing the amount of manual work required by a genealogist. However, due to the high number of false positives, there still exists a need to sort through all record pairs that were classified as a match. The time complexity does become an issue for large datasets but it is still significantly faster than making n^2 comparisons on the entire dataset. In addition, the Cocktail Approach will only be able to perform well if the dataset contains a sufficient amount of true duplicates to analyze and model. Overall, the Cocktail Approach can be used for genealogical record linkage but it does not completely cut out the need for a genealogist to manually sort through the results due to the high number of false positives.

CHAPTER 8

Conclusion

There are several different approaches to finding duplicate records within a dataset. Genealogical records behave differently from records in other domains. Issues such as typographical errors, data entry errors, and missing data are common throughout genealogical records. Many current approaches for genealogical record linkage require a great deal of manual work or training data. The Cocktail Approach has an advantage of not needing any training data and reduces the amount of manual work needed. I re-implemented and modified the Cocktail Approach and tested it on 13 artificially created datasets containing genealogical records. My system obtained an average recall score of 94.6%, an average precision score of 46.4%, and an average f-score of 53.2%. These scores were significantly higher than the results of the baseline system. Overall, the Cocktail Approach has the potential of finding all duplicate records within a genealogical dataset. However it also obtains a high number of false positives. In addition, the performance of the system increases exponentially with large datasets. Nonetheless, the Cocktail Approach does provide a good way to find duplicate genealogical records in situations where training data is not available.

CHAPTER 9

Future Work

There are a number of different ways to improve the Cocktail Approach for genealogical record linkage. One way is to create more specific domain metarules that reflect common patterns in genealogical data for each region and time period. For datasets that were not able to generate a good matching ruleset, predefined domain metarules will provide a baseline for finding duplicates, significantly reducing the number of false positives. This was not done in this thesis because creating domain metarules requires research and a deep understand of the domain, which is outside the scope of this thesis project.

About half of the records within genealogical trees are made up of leaves. In genealogy, the leaf records tend to have more missing information than other records. In addition, the records of women tend to be poorly documented due to unknown maiden names. These poorly documented records greatly reduce the accuracy of the system. A way to increase the accuracy is to take into account the position/place of a record within a family tree and include different confidence levels for matches.

Another way to improve the system is to use different similarity metrics and thresholds for different attributes. Numbers and dates behave differently than string data such as names. In addition, attributes such as first and last names have different properties and behavior than other attributes like address and dates. Using different similarity metrics for numerical data and string data can help to improve the comparison process of attributes. To increase the accuracy

of birth date and death date comparisons, the dates can be simplified to years only. In addition, each attribute should be analyzed to create different similarity thresholds for each attribute. However, this will require a better understanding of each attribute. Statistics obtained from the dataset can be used to generate these similarity thresholds for each attribute.

Structural heterogeneity was one issue not addressed in this thesis. Structural heterogeneity refers to records that have different domain structures. Another issue not addressed is different spellings of names. In many cultures, there are common names that are spelled differently. For example, my name ‘Pooja’ is a fairly common Indian name however, another common spelling of the name is ‘Puja’. In addition, some names are spelled different across different languages. Incorporating solutions to these issues can increase the accuracy of the Cocktail Approach.

BIBLIOGRAPHY

[1] R. Ananthakrishna, S. Chaudhuri, and V. Ganti, “Eliminating Fuzzy Duplicates in Data Warehouses,” in Proceedings of the 28th International Conference on Very Large Data Bases, ser. VLDB ’02. VLDB Endowment, 2002, pp. 586–597.

[2] R. Baxter, P. Christen, and T. Churches, “A Comparison of Fast Blocking Methods for Record Linkage,” in KDD 2003 WORKSHOPS, 2003, pp. 25–27.

[3] O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S. E. Whang, and J. Widom, “Swoosh: A Generic Approach to Entity Resolution,” The VLDB Journal, vol. 18, no. 1, pp. 255–276, Jan. 2009.

[4] B. Cookson, K. Boyer, J. Collings, J. Hamilton, K. Jefferson, G. Parkinson, D. Reid, D. Thayne, and M. Wolfgramm, “Correlating Genealogy Records Systems and Methods,” July 24 2007, US Patent 7,249,129.

[5] A. P. Dempster, N. M. Laird, and D. B. Rubin, “Maximum Likelihood from Incomplete Data via the EM Algorithm,” Journal of the Royal Statistical Society, Series B, vol. 39, no. 1, pp. 1–38, 1977.

[6] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, “Duplicate Record Detection: A survey,” IEEE Trans. on Knowl. and Data Eng., vol. 19, no. 1, pp. 1–16, Jan. 2007.

[7] I. P. Fellegi and A. B. Sunter, “A Theory for Record Linkage,” Journal of the American Statistical Association, vol. 64, pp. 1183–1210, 1969.

[8] K. Goiser and P. Christen, “Towards Automated Record Linkage,” in Proceedings of the Fifth Australasian Conference on Data Mining and Analystics - Volume 61, ser. AusDM ’06. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2006, pp. 23–31.

[9] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava, “Approximate String Joins in a Database (Almost) for Free,” in Proceedings of the 27th International Conference on Very Large Data Bases, ser. VLDB ’01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 491–500.

[10] M. A. Hernandez and S. J. Stolfo, “Real-World Data is Dirty: Data Cleansing and the Merge/Purge Problem,” Data Min. Knowl. Discov., vol. 2, no. 1, pp. 9–37, Jan. 1998.

[11] T. H. Herzog, F. Scheuren, and W. E. Winkler, “Record Linkage,” Wiley Interdisciplinary Reviews: Computational Statistics, vol. 2, no. 5, pp. 535–543, 2010.

[12] M. A. Jaro, “Advances in Record-Linkage Methodology as Applied to Matching the 1985 Census of Tampa, Florida,” Journal of the American Statistical Association, vol. 84, no. 406, pp. pp. 414–420, 1989.

[13] J. Lawson, D. White, B. Price, and R. Yamagata, “Probabilistic Record Linkage for Genealogical Research.” BYU Studies 41.

[14] S. B. Needleman and C. D. Wunsch, “A General Method Applicable to the Search for Similarities in The Amino Acid Sequence of Two

Proteins," Journal of Molecular Biology, vol. 48, no. 3, pp. 443 – 453, 1970.

[15] H. B. Newcombe, "Record Linking: The Design of Efficient Systems for Linking Records into Individual and Family Histories," American Journal of Human Genetics, vol. 19, no. 3, pp. 335–359, May 1967.

[16] H. B. Newcombe, J. M. Kennedy, S. Axford, and A. James, "Automatic Linkage of Vital Records," Science, vol. 130, no. 3381, pp. 954–959, October 1959.

[17] S. A. P. Ong, "A Comparative Study of Record Matching Algorithms," (Master Thesis, RWTH Aachen, Germany and University of Edinburgh, 2008). Scotland: Germany and University of Edinburgh (2008).

[18] A. Onisko, P. J. F. Lucas, and M. J. Druzdzel, "Comparison of Rule-Based and Bayesian Network Approaches in Medical Diagnostic Systems," in Proceedings of the 8th Conference on AI in Medicine in Europe: Artificial Intelligence Medicine, ser. AIME '01. London, UK: SpringerVerlag, 2001, pp. 283–292.

[19] E. Rahm and H. H. Do, "Data cleaning: Problems and Current Approaches," IEEE Data Engineering Bulletin, vol. 23, 2000.

[20] S. Tejada, C. A. Knoblock, and S. Minton, "Learning object identification rules for information integration," Inf. Syst., vol. 26, no. 8, pp. 607–633, Dec. 2001.

[21] D. R. Wilson, “Graph-based Remerging of Genealogical Databases,” In Proceedings of the Workshop on Technology for Family History and Genealogical Research (FHT 2001), pp. 38-39, 2001.

[22] W. E. Winkler, “Improved Decision Rules in the Fellegi-Sunter Model of Record Linkage,” In *Technical Report Statistical Research Report Series RR93/12, US Bureau of the Census, Washington*, 1993.

[23] W. E. Winkler, “Matching and Record Linkage,” in Business Survey Methods. Wiley, 1995, pp. 355–384.

[24] W. E. Winkler, “Machine Learning, Information Retrieval, and Record Linkage,” in Proceedings of the Survey Research Methods Section, American Statistical Association, 2000.

[25] W. E. Winkler, “Methods for Record Linkage and Bayesian Networks,” U.S. Bureau of the Census, Washington, D.C., Tech. Rep. Statistical Research Report Series RRS2002/05, 2002.

[26] W. E. Winkler, “Overview of Record Linkage and Current Research Directions,” in Statistical Research Division, U.S. Census Bureau, 2006, p. 44.

[27] W. E. Winkler and Y. Thibaudeau, “An Application of the Gellegi-Sunter Model of Record Linkage to the 1990 U.S. Decennial Census,” in U.S. Decennial Census”. Technical report, US Bureau of the Census, 1987.

APPENDIX A

Generated Rulesets

The generated equational theory ruleset generated for each dataset by my system is listed below. For each *rule*, or vector, if the *i-th* index is 1, then the attributes at that index must match. The index-attribute mapping is presented below:

```
FIRST_NAME  = 0
MIDDLE_NAME = 1
LAST_NAME   = 2
BIRTH_DATE  = 3
DEATH_DATE  = 4
BURIAL      = 5
GENDER      = 6
ANCESTOR    = 7
NAT_PARENTS = 8
PARENTS     = 9
MARRIAGE    = 10
```

Dataset 1:

RULE_SET: [[1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0], [1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1]]

Dataset 2:

RULE_SET: [[1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0], [1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1], [1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1], [1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0]]

Dataset 3:

RULE_SET: [[1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1], [1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1], [0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0], [1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0], [1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0], [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0], [1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0], [1, 0, 1, 0, 0, 0, 0, 0, 0, 1,

1], [0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0], [1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1], [1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1], [1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1], [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1], [1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1], [1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1], [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1], [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0], [1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1], [1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1], [1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1], [1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1], [0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0], [0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0], [1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0], [0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1], [0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1], [0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0], [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1], [1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1], [1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1], [1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1], [1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0], [1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0], [0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1], [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1], [0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1], [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0], [1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1], [0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0], [1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0], [1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0], [0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0], [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0], [1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1], [1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0], [0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1], [1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1], [0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1], [0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1], [1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0], [0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0]]

Dataset 4:

RULE_SET: [[1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1], [1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0], [1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1], [0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0]]

Dataset 5:

RULE_SET: [[0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1], [1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0], [1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1], [1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1], [1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1], [1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1]]

Dataset 6:

RULE_SET: [[1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1], [1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0], [1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1], [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0], [1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1], [1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1], [0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1], [1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1], [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1], [0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0], [0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0], [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0], [1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1], [0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0]]

Dataset 7:

RULE_SET: [[1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1], [1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1]]

Dataset 8:

RULE_SET: [[1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1], [1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0], [1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0], [0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1], [0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0], [1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0], [1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0], [1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1], [1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1], [1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0], [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1], [1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1], [1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1], [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0], [1, 0, 1, 0,

0, 0, 1, 0, 0, 0, 1], [0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1], [0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0], [1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1], [0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1], [0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0], [0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1], [1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0], [1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0], [1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0], [1, 0, 1, 0, 0, 0, 0, 0, 1, 1], [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1], [1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1], [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1], [0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1], [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0], [0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0], [1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0], [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0], [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1], [1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1], [0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1], [1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0]]

Dataset 9:

RULE_SET: [[1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1], [1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0], [1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1], [1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1], [0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1]]

Dataset 10:

RULE_SET: [[0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1], [1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1], [1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1], [1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0], [1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1], [0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0], [1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0]]

Dataset 11:

RULE_SET: [[1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0], [1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1], [0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0]]

Dataset 12:

RULE_SET: [[1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1], [1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1], [1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1], [0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1], [1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1], [0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0], [1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0]]

Dataset 13:

RULE_SET: [[1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0], [0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0], [1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0], [1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0], [1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0], [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]]