A FRAMEWORK FOR ACCURATE, EFFICIENT PRIVATE RECORD LINKAGE

by

Elizabeth Ashley Durham

Dissertation

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

in

Biomedical Informatics

May, 2012

Nashville, Tennessee

Dedicated

to

the ones I love

ACKNOWLEDGEMENTS

A great number of people have been instrumental to the completion of this work, and I'd like to thank a few of them now.

I would like to first thank the U.S. National Library of Medicine (grant 2-T15LM07450-06), the U.S. National Institutes of Health (grant R01 LM009989), and the Team for Research in Ubiquitous Secure Technology (grant CCF-0424422) for the funding that made this work possible.

I would like to acknowledge my advisor and mentor, Brad Malin, whom I have been incredibly fortunate to study under. I have a sneaking suspicion I may be disappointed in every future boss because I've already had the best boss possible. I am thankful for the countless hours Brad has spent reading through and improving my writing, tossing around ideas, and steering me in the right direction. I hope to some day be half as smart as he is.

I have been fortunate to work within a team of brilliant people with synergistic and complementary skills. Thanks to Yuan Xue, Murat Kantarcioglu, Csaba Toth, and Mehmet Kuzu for your time, ideas, feedback, and encouragement. Also, many thanks to the members of the Health Information Laboratory for their support and feedback.

I would like to thank my committee members – Brad Malin, Yuan Xue, Murat Kantarcioglu, Mark Frisse and Dario Giuse – for their time, support, and constructive feedback throughout this process.

I am honored to have participated in the excellent training program at Vanderbilt University's Department of Biomedical Informatics. The department is incredibly supportive of students and their research interests. This is due in no small part to Cindy Gadd, the exceptional Director of Graduate Studies. And behind the scenes, Rischelle Jenkins is the one keeping the wheels on the whole operation, smiling all the while and bringing a smile to the face of everyone she meets.

I thank all of my friends who have given up a friend over these last few months when

I've been tucked away writing this dissertation. And finally, I'd like to thank my family who have been immensely supportive and encouraging at each step along the way.

Thank you all.

“I am so clever that sometimes I don’t understand a single word of what I am saying.”

- Oscar Wilde

ABSTRACT

Record linkage is the task of identifying records from multiple data sources that refer to the same individual. Private record linkage (PRL) is a variant of the task in which data holders wish to perform linkage without revealing identifiers associated with the records. PRL is desirable in various domains, including health care, where it may not be possible to reveal an individual's identity due to confidentiality requirements. In medicine, PRL can be applied when datasets from multiple care providers are aggregated for biomedical research, thus enriching data quality by reducing duplicate and fragmented information. Additionally, PRL has the potential to improve patient care and minimize the costs associated with replicated services, by bringing together all of a patient's information.

This dissertation is the first to address the entire life cycle of PRL and introduces a framework for its design and application in practice. Additionally, it addresses how PRL relates to policies that govern the use of medical data, such as the HIPAA Privacy Rule. To accomplish these goals, the framework addresses three crucial and competing aspects of PRL: 1) computational complexity, 2) accuracy, and 3) security. As such, this dissertation is divided into several parts. First, the dissertation begins with an evaluation of current approaches for encoding data for PRL and identifies a Bloom filter-based approach that provides a good balance of these competing aspects. However, such encodings may reveal information when subject to cryptanalysis and so, second, the dissertation presents a refinement of the encoding strategy to mitigate vulnerability without sacrificing linkage accuracy. Third, this dissertation introduces a method to significantly reduce the number of record pair comparisons required, and thus computational complexity, for PRL via the application of locality-sensitive hash functions. Finally, this dissertation reports on an extensive evaluation of the combined application of these methods with real datasets, which illustrates that they outperform existing approaches.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOTATION

A	the dataset belonging to data holder Alice
B	the dataset belonging to data holder Bob
C	the number of unique trigrams
D	the difference vector for the *Trigrams* private string comparator
E	an encoding function
F	used to define locality-sensitive hash function families
G	a locality-sensitive hash family
H	entropy
I	used to illustrate the Jaccard coefficient and Hamming distance
J	used to illustrate the Jaccard coefficient and Hamming distance
L	$\|A \times B\|$
M	the class of Matches
N	used to illustrate entropy
O	used to illustrate entropy
P	probability mass function
R	Regenstrief record identifier
S	$A \cup B$
T	the threshold for the *Trigrams* private string comparator
U	the class of Non-matches
V	Vanderbilt record identifier
W	used to define locality-sensitive hash function families
X	used to illustrate entropy
Y	used to illustrate entropy
a	a record in dataset A
b	a record in dataset B

c	the number of clusters in c-means clustering
d	the Dice coefficient
e	the number of record pairs output from a blocking algorithm
f	the number of fields in a record
g	the number of n-grams used in Bloom filter encoding
h	a hash function
i	used to index fields in a record
j	an element in embedding reference strings for the *Embedding* private string comparator
k	the number of hash functions used to in Bloom filter encoding
l	Jaro-Winkler string comparator prefix length
m	the number of bits in a Bloom filter
n	the size of tokens used to create n-grams
o	used to illustrate entropy
p	the probability with which a bit in Bloom filter is set
q	a security constraint used in record-level Bloom filter encodings
r	used in the r-tree data structure
s	a string in the reference space of the *Embedding* private string comparator
t	used to illustrate entropy
u	used to illustrate entropy
v	a scaling factor used to illustrate Jaro distance
w	field weight
x	used to index records in dataset A
y	used to index records in dataset B
z	dataset number
α	a Bloom filter encoding
β	a Bloom filter encoding
γ	a vector containing the field similarity values for a record pair

δ	an ID used to describe a computer used for experiments
ϵ	an ID used to describe a computer used for experiments
ζ	an ID used to describe a computer used for experiments
θ	an ID used to describe a computer used for experiments
ι	an ID used to describe a computer used for experiments
λ	the expected number of *True Matches*
μ	the number of iterations, or times a locality-sensitive hash function family is applied
ν	the number of fields utilized in iterative token blocking
ξ	used to illustrate linkage of more than two datasets
π	a generic Bloom filter encoding
ρ	used to index bits in a Bloom filter encoding
σ	used to illustrate *F-score*
τ	used to illustrate *F-score*
ϕ	the number of hash functions applied in a locality-sensitive hash function family
χ	used to illustrate Jaro distance
ψ	used to illustrate Jaro distance
ω	used to illustrate entropy
$\mid \bullet \mid$	the number of elements in set $\bullet$
$\mid A \mid$	the number of records in dataset A
$\mid B \mid$	the number of records in dataset B
c_1	used to illustrate Jaro distance
c_2	used to illustrate Jaro distance
$CG1$	an experimental method evaluated in Chapter 6
$CG2$	an experimental method evaluated in Chapter 6
$CG3$	an experimental method evaluated in Chapter 6
$CG4$	a control method evaluated in Chapter 6
$CG5$	a control method evaluated in Chapter 6

d_j	the Jaro distance
d_{jw}	the Jaro-Winkler distance
e_{tp}	the number of true positives output by a given blocking algorithm
f_1	an element drawn from the set F in illustrating locality-sensitive hash function families
f_2	an element drawn from the set F in illustrating locality-sensitive hash function families
k_1	a key used in hashing
k_2	a key used in hashing
L_{tp}	the number of true matches
$m[i]$	the probability that field i agrees amongst *True Matches*
m_{FBF}	the number of bits in a field-level Bloom filter
m_{RBF}	the number of bits in a record-level Bloom filter
p_1	probability $> p_2$ used to define locality-sensitive hash function families
p_2	distance $< p_1$ used to define locality-sensitive hash function families
r_1	distance $< r_2$used to define locality-sensitive hash function families
r_2	distance $> r_1$ used to define locality-sensitive hash function families
s_1	used to illustrate Jaro distance
s_2	used to illustrate Jaro distance
t_1	the “loose” threshold in the Canopy blocking approach
t_2	the “tight” threshold in the Canopy blocking approach
$w[i]$	the weight associated with field i
$w_a[i]$	the agreement weight associated with field i
$w_d[i]$	the disagreement weight associated with field i
$u[i]$	the probability that field i agrees amongst *True Non-matches*
☼	used to illustrate linkage of more than two datasets
Alice	the data holder for dataset A
Bob	the data holder for dataset B
Charlie	the third party who performs record linkage

Dan	an additional third party who can help in parameterization
ARRA	American Recovery and Reinvestment Act
CSP	constraint satisfaction problem
DB	database
DOB	date of birth
EM	Expectation Maximization
FBF	field-level Bloom filter
FDA	Food and Drug Administration
FIPS	Federal Information Processing Standards
FN	false negative
FP	false positive
FS	Fellegi and Sunter
HIPAA	Health Information Portability and Accountability Act
HITECH	Health Information Technology for Economic and Clinical Health
HLSH	Hamming-based locality-sensitive hashing
HMAC	hash message authentication code
ID	identifier
JLSH	Jaccard-based locality-sensitive hashing
JW	Jaro-Winkler string comparator
LSH	locality-sensitive hash
MI	mutual information
NIH	National Institutes of Health
NIST	National Institute of Standards and Technology
RBF	record-level Bloom filter
RR	reduction ratio
PB	private blocking
PC	pairs completeness

PHI	protected health information
PRL	private record linkage
PSC	private string comparator
RBF	record-level Bloom filter
SSN	Social security number
TP	true positive
TN	true negative
UPI	universal patient identifier
U.S.	United States

CHAPTER I

INTRODUCTION

1.1 Overview

Record linkage is the task of identifying records from multiple data sources that refer to the same individual. For example, in the medical domain, record linkage can be applied to patient records held by various clinical care providers in order to identify records that refer to the same patient. The primary benefit of record linkage is that it enables a more complete picture of an individual by bringing together his information.

Record linkage in its most basic form, however, is not always possible due to regulatory constraints and social concerns about personal privacy [113]. For example, data sharing policies, such as the Health Insurance Portability and Accountability Act (HIPAA) Privacy Rule, prohibit sharing identifying patient information in some cases [112]. In light of these concerns and restrictions, private record linkage (PRL) approaches have been developed, based on the use of encoded values. PRL enables data holders to identify records referring to the same individuals without revealing identifying information about the individuals [34].

1.2 Applications of Private Record Linkage

When analyzing a dataset resulting from the aggregation of data from multiple sources, record linkage enriches data quality, prevents duplication, and ultimately enables more accurate analysis [56]. If record linkage is not performed, the aggregated dataset may contain multiple records corresponding to the same individual.

PRL can enhance data analysis in many domains. Several examples, both within and beyond the field of medicine, are provided below.

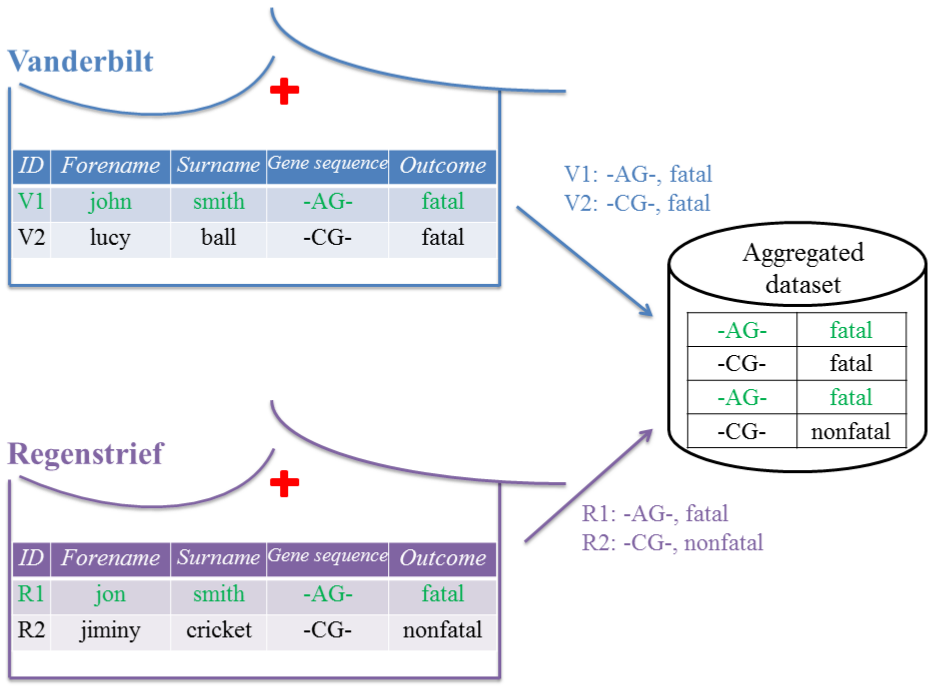
1.2.1 PRL as a Precursor to De-identified Data Sharing for Research

The medical domain is of particular interest to this dissertation because sharing patient information is critical for biomedical research [98]. The U.S. National Institutes of Health (NIH), for instance, requires that patient information used in federally funded research studies be released for research only in a form "free of identifiers that would permit linkages to individual research participants and variables that could lead to deductive disclosure of the identity of individual subjects" [114]. In other words, information that could identify an individual should not be shared.

For a practical example, imagine that a life science researcher wishes to conduct a study on the relationship between two distinct biomedical attributes (*e.g.*, genetic sequence and disease status) and needs to use information from multiple data sources to meet a minimal sample size for statistical significance. A patient may visit multiple health care providers, where his information is collected and applied in distinct studies [84]. If the patient's information is duplicated across providers, and this information is then aggregated, queries issued on the resulting dataset may overestimate correlations. Consider the example shown in Figure 1.1(a). Only one individual has the genetic sequence "–AG–" and fatal outcome, but the query result from the aggregated dataset indicates two individuals harbor this variant.

On the other hand, if a patient's information is fragmented, such that a particular attribute (*e.g.*, genetic sequence) is submitted from one provider and a separate attribute (*e.g.*, outcome) is submitted by another provider, the query results would be an underestimate of the correlation of the two attributes. Consider the example shown in Figure 1.1(b). One individual has the genetic sequence "–AG–" and fatal outcome, but the query result from the aggregated dataset indicates no individuals have this sequence and outcome.

However, PRL can be applied as a precursor to de-identified data sharing to mitigate bias in such statistical analyses – without requiring revelation of identifying information. For an example, see Figure 1.2.

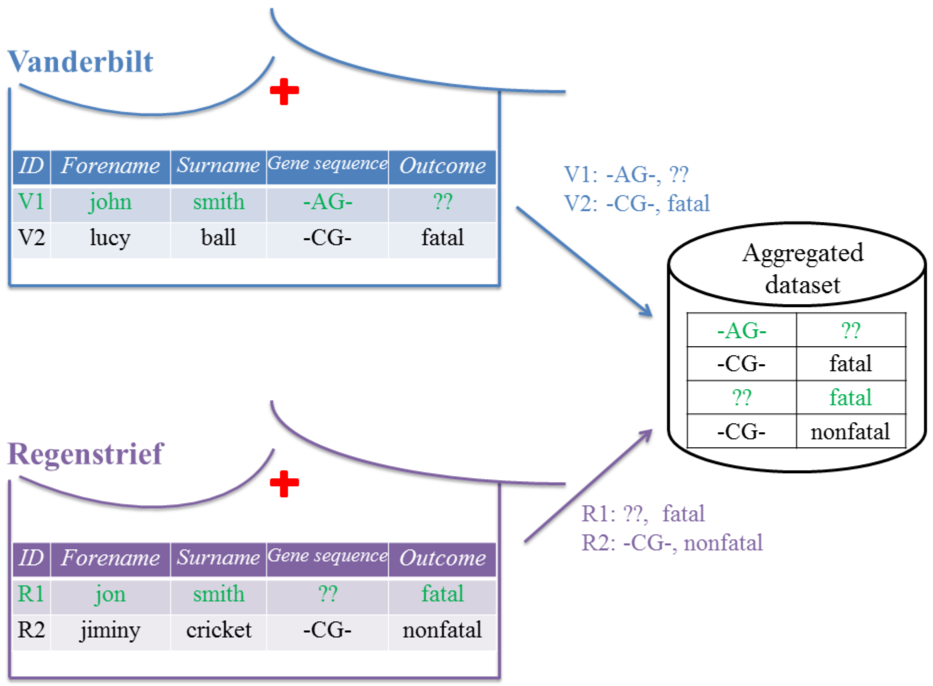


ID	Forename	Surname	Gene sequence	Outcome
V1	john	smith	-AG-	fatal
V2	lucy	ball	-CG-	fatal

-AG-	fatal
-CG-	fatal
-AG-	fatal
-CG-	nonfatal

ID	Forename	Surname	Gene sequence	Outcome
R1	jon	smith	-AG-	fatal
R2	jiminy	cricket	-CG-	nonfatal

(a) Duplicated data can lead to overestimates of correlation.



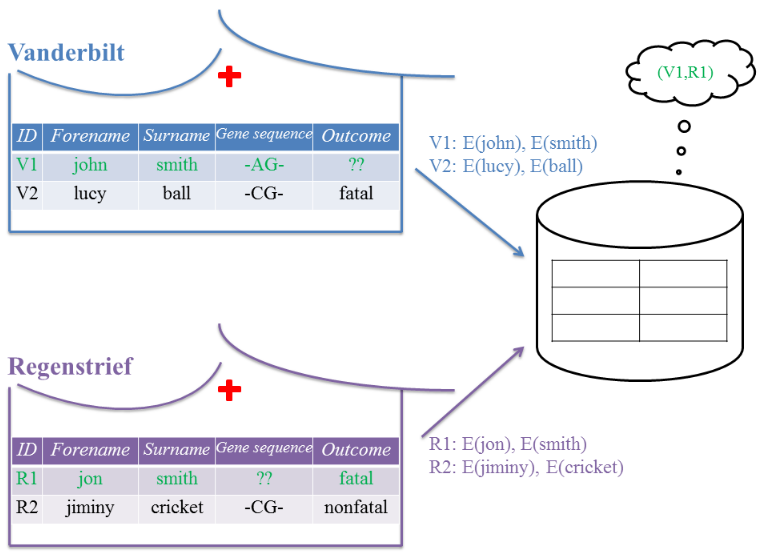
ID	Forename	Surname	Gene sequence	Outcome
V1	john	smith	-AG-	??
V2	lucy	ball	-CG-	fatal

-AG-	??
-CG-	fatal
??	fatal
-CG-	nonfatal

ID	Forename	Surname	Gene sequence	Outcome
R1	jon	smith	??	fatal
R2	jiminy	cricket	-CG-	nonfatal

(b) Fragemented data can lead to underestimates of correlation.

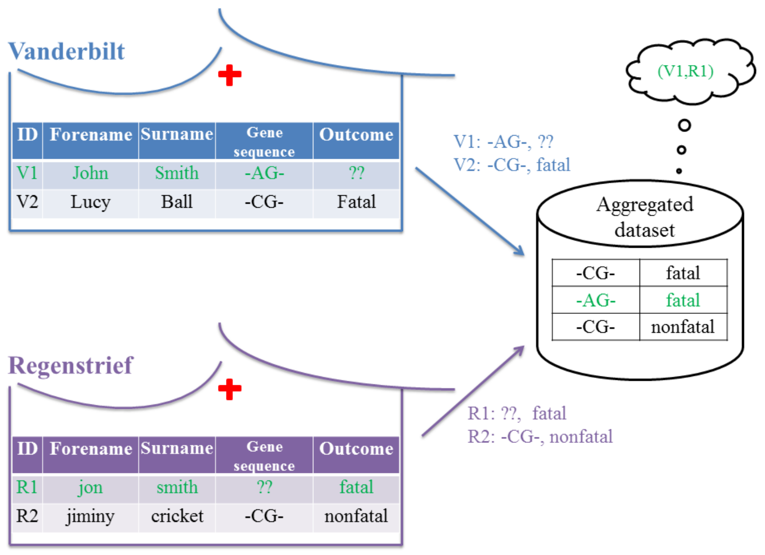
Figure 1.1: Problems with aggregating data without performing record linkage. The records referring to the same individual are shown in green.



ID	Forename	Surname	Gene sequence	Outcome
V1	john	smith	-AG-	??
V2	lucy	ball	-CG-	fatal

ID	Forename	Surname	Gene sequence	Outcome
R1	jon	smith	??	fatal
R2	jiminy	cricket	-CG-	nonfatal

(a) Step 1: Private record linkage. Encoded identifiers (where "E" indicates an encoding function) are sent to the centralized repository, where record linkage is performed. A note is made that records V1 and R1 refer to the same individual.



ID	Forename	Surname	Gene sequence	Outcome
V1	John	Smith	-AG-	??
V2	Lucy	Ball	-CG-	Fatal

-CG-	fatal
-AG-	fatal
-CG-	nonfatal

ID	Forename	Surname	Gene sequence	Outcome
R1	jon	smith	??	fatal
R2	jiminy	cricket	-CG-	nonfatal

(b) Step 2: De-identified data sharing. The de-identified data is then sent to the centralized repository, along with the record ID, so that records identified in the previous step as referring to the same individual, can then be merged.

Figure 1.2: PRL can be applied as a precursor to de-identified data sharing, to enable more accurate data analysis. The records referring to the same individual are shown in green.

1.2.2 PRL for Improving Medical Care

The decentralized nature of health care systems means that a patient's medical information is stored in a fragmented form across multiple health care providers. A more complete view of a patient's medical information can be realized by bringing together the separate pieces of information held by providers. This has the potential both to increase patient safety and decrease the cost of medical care by allowing a care provider to determine if certain examinations have already been rendered, thus minimizing replication of services.

A universal patient identifier (UPI), a unique number assigned to each patient and shared across providers, has been proposed for use in the United States health care system and has been the topic of much debate [11, 28, 57, 106, 122]. While such an identifier, if recorded consistently, would provide a straightforward means for aggregating a patient's information, this idea has met opposition on many fronts.

One objection to a UPI is the challenge of administering it on a large scale. For example, consider another number that is intended to be a unique identifier – the Social security number (SSN). It has been reported that more than 20 million people have been assigned more than one SSN, and more than 40 million SSNs are associated with multiple individuals [13].

Even if a unique number is assigned to each patient, missing data and data entry errors can present a challenge in identifying all of a patient's records. In one study, a linkage of two medical datasets using SSN was performed as part of the Shared Pathology Information Network project at the Regenstrief Institute for Health Care [51]. In this study, 35% of the records lacked SSNs and were therefore unable to be linked. Amongst records where SSNs were present, the value of each SSN was recorded incorrectly in up to 9.2% of the records. This demonstrates that even a supposedly unique identifier is not necessarily a magic bullet that will provide for an efficient and accurate way to aggregate all of the records corresponding to an individual.

In addition to the administrative challenges associated with a UPI, there are other social objections to human enumeration. Notably, a 1973 U.S. Department of Health and Human Services report rejected any move toward “Standard Universal Identifiers” on confidentiality grounds [28]. Therefore, due to social concerns and the administrative burden, the use of UPIs has not come to fruition in the U.S. health care system.

To summarize, in the absence of a consistently recorded universal patient identifier, PRL can add great value in medical care. A more complete view of a patient’s medical information, afforded by record linkage, has the potential both to increase patient safety and decrease the cost of medical care by minimizing replication of services.

1.2.3 PRL for Public Health Surveillance

PRL can also be applied to strengthen surveillance activities in the context of public health. For example, the U.S. Food and Drug Administration (FDA) intends to monitor the safety of drugs and FDA-regulated medical products in real time, rather than retrospectively. A key aim of the FDA’s recently launched project, the Sentinel Initiative, is to develop a fully operational active surveillance system for monitoring the safety of FDA-regulated medical products [3]. The initiative will integrate information from electronic medical records with existing methods for adverse event detection and post-market surveillance [117]. PRL has the potential to aid the FDA’s efforts to monitor, in real time, the safety of drugs and medical products by integrating information from multiple sources.

1.2.4 PRL for Monitoring Participation in Drug Trials

A clinical trial is a study commonly used in pharmaceutical development to determine the safety and efficacy of a drug. Patients with a condition the drug is intended to treat are recruited for participation in the trial. An important requirement is that participants take part in only a single trial at a time; if a patient were to participate in multiple trials and have an adverse effect, it would be unclear which drug caused the effect.

Clinical trials often compensate patients for their participation; however, this leads some patients to participate in multiple trials in order to receive compensation for each [6]. As trials are covered by the HIPAA Privacy Rule [112] and cannot release identifying information about trial participants, detecting these "professional guinea pigs" can be difficult. However, PRL provides a tool which trial administrators can use to detect and exclude patients participating in multiple trials, thus facilitating more accurate pharmaceutical research.

1.2.5 PRL Outside of the Medical Domain

PRL also has many applications outside of the medical domain. For an example, consider PRL in the context of counter-terrorism efforts. When a terrorist suspect is identified by an intelligence agency, PRL can be applied to identify records from multiple data sources that cannot freely share identifying information. The intelligence agency can then detect aliases or combine information about the individual to learn about his actions or co-conspirators [37]. For example, the European Union recently started to share information about inbound travelers (*i.e.*, Passenger Name Records) to U.S. officials for inspection against terrorist "watchlists" [73]. These actions have met significant opposition due to concerns over personal privacy and autonomy [47]. However, the adoption of PRL could alleviate this tension because it could enable a protocol in which only records referring to suspects on watchlists would be revealed [75].

1.3 Contributions of this Work and its Potential Impact on the Field of Medicine

Now that we have considered several applications of PRL both within and beyond the field of medicine, we examine the specific contributions of this research and consider its potential impact on the field of medicine. The sharing of de-identified patient information for research is a common practice in the medical field because it preserves clinically-relevant information while complying with data sharing policies set forth in HIPAA. While de-identified data is useful for many research applications, performing record linkage on de-identified data is not ideal because the goals of record linkage and de-identification are fundamentally at odds with one another. The goal of de-identification is to "unlink" an individual's record from his identity. The goal of record linkage, in contrast, is not only to link one record to an individual, but to link all of his corresponding records[*]. Clearly, these goals are in conflict, and de-identified data is ill-suited for record linkage.

A common method for performing PRL is to encode patient records such that the contents are indecipherable but can still be compared to one another in order to estimate the similarity of records. However, the sharing of encoded data for PRL has not yet become widely accepted in the field of medicine. This is due, in part, to several open questions about the feasibility and security of PRL. Specific open research questions are:

1. Can PRL be performed securely without revealing sensitive patient information?
2. Can highly accurate record linkage be achieved when records are encoded?
3. How well does PRL scale for the large datasets required for medical research?

The research described in this dissertation focuses on these core research questions and makes the following specific contributions:

1. **Evaluation of proposed encoding techniques for use in PRL.**

   While many encoding techniques have been presented in the literature, the extent to which they protect patient information, and their accuracy with respect to one another, has not been quantified on a common scale. Chapter 3 describes an evaluation of private string comparators (PSCs) in terms of correctness, computational complexity, and security. We identify one of the encoding techniques as particularly promising for use in PRL. However, additional work revealed a security vulnerability of this encoding technique. Specifically, work performed jointly with collaborators

[*]Note that while the goal of PRL is to link all records corresponding to an individual together, it is not necessary to reveal the individual's identity. In fact, the explicit goal of PRL is to bring together all records corresponding to an individual *without* revealing his identity.

demonstrated that, for given parameter settings, a frequency-based cryptanalysis attack can reveal some plaintext values [76]. Building upon these results, a novel, more robust encoding that addresses the security concern is described in Chapter 4.

2. **Incorporation of a field weighting technique into a secure encoding method.** Different data fields have different discriminatory powers in resolving identity (*e.g.*, *Forename* is more useful in discriminating between individuals than *Gender*), and an accurate record linkage algorithm will leverage this information. Chapter 4 introduces a method to incorporate field weights into an encoding technique in order to account for the varying discriminatory powers of data fields. This provides a means for assigning greater weight to more discriminatory fields, thus facilitating highly accurate record linkage.

3. **Development of a method that reduces the computational complexity of PRL.**
Record linkage is a computationally intensive process and, in practice, its sheer computational complexity is a barrier to its implementation. The matter of scalability is addressed in Chapter 5, and a novel algorithm that reduces the computational complexity of record linkage is developed, presented, and evaluated. This algorithm demonstrates that PRL can be performed efficiently, and accurately, on very large datasets.

The answers to these questions pave the way for a robust, real-world PRL application. This PRL framework is presented and evaluated in Chapter 6.

1.4 Private Record Linkage and Policy

While the methods described in this dissertation will demonstrate that PRL is feasible and secure from a technological standpoint, the policies governing this technology are equally important. If sharing an encoded form of identifying information is not permissible with respect to organizational or legal policies, the fact that it is technologically possible ceases to be relvant. Therefore, the implications for PRL of two key documents that govern the use of medical data, HIPAA and the Health Information Technology for Economic and Clinical Health (HITECH) Act of the American Recovery and Reinvestment Act (ARRA) stimulus, are examined in Section 7.4 of this dissertation.

1.5 Dissertation Outline

In this chapter, we introduced the general idea of record linkage, a special case in the form of PRL, and several applications both within and beyond the field of medicine.

The remainder of this dissertation is organized as follows. Chapter 2 presents the steps involved in the record linkage process and describes the datasets and implementation details used throughout this dissertation. Chapter 3 evaluates PSCs to determine the state of the art and to quantify the accuracy, security, and computational complexity of existing methods on a common ground. This PSC evaluation identifies security concerns for existing encoding techniques, which are addressed by work described in Chapter 4 to develop a novel, more secure encoding technique. Chapter 5 presents a method for reducing the computational complexity of PRL and therefore improving its scalability. Chapter 6 brings together the work of the previous chapters and presents an evaluation of the entire PRL framework. Finally, Chapter 7 summarizes the dissertation and its implications while discussing research questions that are left to future work.

CHAPTER II

BACKGROUND AND EVALUATION SETTINGS

This chapter details the steps involved in record linkage. Additionally, it describes the framework, implementation details, and datasets used throughout this research.

2.1 Steps Involved in Record Linkage

Record linkage, the task of identifying records that refer to the same individual, can be viewed as a function where the inputs are sets of records, and the output is the set of *Predicted Matches* – record pairs predicted to refer to the same individual – and *Predicted Non-matches* – record pairs predicted not to refer to the same individual. The record linkage function can be decomposed into several distinct steps. The steps central to the record linkage process are described in detail in Sections 2.1.1 through 2.1.8 and are visually shown in Figure 2.1.

While a comprehensive list of the notation used throughout this dissertation can be found on page xxii, highlights are as follows. We consider the linkage of records from two data holders, who we call Alice and Bob. Each record is comprised of f fields. A denotes the set of records provided by data holder Alice, $|A|$ is the number of records in dataset A, a_x indicates the x^{th} record within set A, and $a_x[i]$ refers to the value of the i^{th} field in record a_x, where $i \in \{1, \dots ,f\}$. Similarly, B, $|B|$, b_y, and $b_y[i]$ correspond to the set of records provided by data holder Bob. The goal of record linkage, then, is to correctly classify all record pairs, the space of the Cartesian product of A and B, into the classes M (*Predicted Matches*) and U (*Predicted Non-matches*) such that the record pairs in the class *Predicted Matches* refer to the same individual (*i.e.*, are *True Matches*) and the record pairs in the class *Predicted Non-matches* do not refer to the same individual (*i.e.*, are *True Non-matches*).

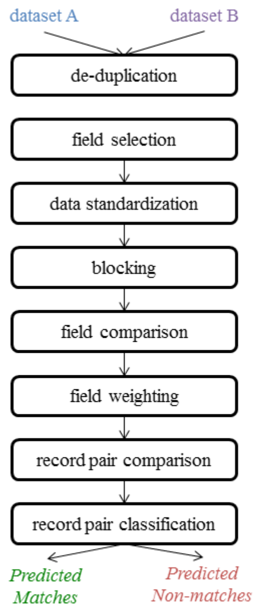


Figure 2.1: A sketch of the current record linkage process.

2.1.1 De-duplication

Record linkage becomes more straightforward if each individual is represented by only a single record *within* each dataset. Therefore, the first step in record linkage, de-duplication, is often an internal record linkage within each dataset to remove duplicate records.

2.1.2 Field Selection

In this step, a set of fields describing characteristics of an individual (*e.g.*, *Gender*, *Date of Birth* (*DOB*), *Forename*, *etc.*), is selected for use in record linkage. In the absence of a unique identifier, quasi-identifiers can be used. The term "quasi-identifier" was first coined by Dalenius [38] and refers to fields that reveal some information about an individual, but not necessarily enough to uniquely identify an individual. For example, *DOB* and *Gender* are quasi-identifiers. While no single quasi-identifier can necessarily uniquely identify an individual, when considered in combination, a set of quasi-identifying attributes may be able to uniquely identify an individual. It is important that, to the extent possible, the set of fields selected is sufficient for uniquely identifying an individual* (*i.e.*, records should have the same values for all fields only if they refer to the same individual).

Additionally, one should choose fields for which the records contain data of high quality. For example, imagine that the set of quasi-identifiers {*Forename*, *Surname*, *Gender*, *DOB*, *Ethnicity*, *Telephone number*} were, in combination, able to uniquely identify an individual. However, suppose that *Forename* was not recorded for half of the records in the datasets to be linked. In this case, additional fields may be needed to ensure that the uniqueness property holds for the selected fields.

2.1.3 Data Standardization

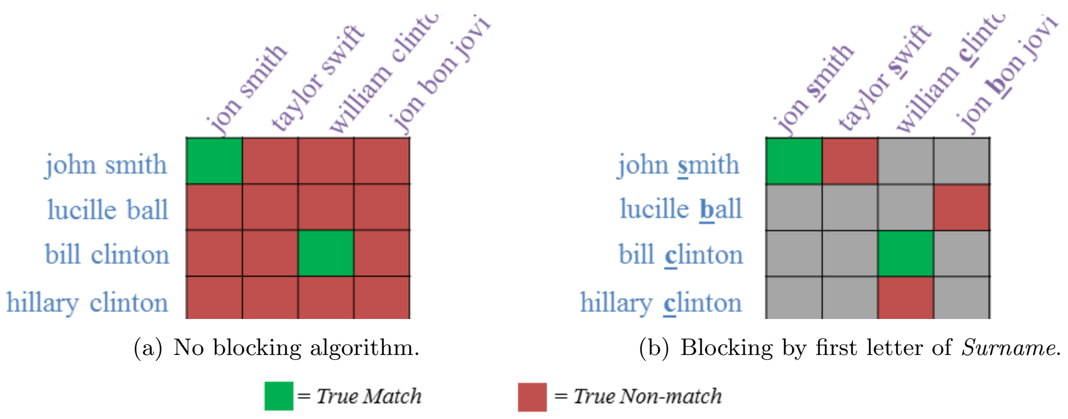
The data holders involved in the linkage should agree upon a set of standardization and pre-processing rules. The manner in which data values are recorded must be standardized across all parties if accurate record linkage is to be performed. For example, it is not suitable for one party to encode *Gender* as “male” and “female” while another party encodes gender as “M” and “F”. Examples of pre-processing rules are that all text should be converted to the same case, hyphens should be removed, *etc.*

While data standardization is imperative for accurate record linkage, it is not the focus of this work. For additional information on data standardization, please see the work by Herzog [56].

2.1.4 Blocking

When two datasets, A and B, are linked, each record in dataset A must be compared to each record in dataset B. To reduce the number of record pair comparisons required, a technique called blocking is often used, in which record pairs unlikely to be *True Matches* are removed from consideration. Consider the example shown in Figure 2.2(a). In this example, datasets A and B each contain four records (*i.e.*, $|A| = |B| = 4$). In order to identify the two true

*In some cases, it may not be possible to acquire enough fields to distinguish between two people. An example of this situation is a dataset that contains records corresponding to twins, between whom it is often difficult to differentiate in record linkage applications.



(a) No blocking algorithm.

(b) Blocking by first letter of *Surname*.

Figure 2.2: a) Each record from dataset A, shown in blue, is compared with each record from dataset B, shown in purple, and the record pairs are then classified. When a blocking algorithm is not utilized, 16 record pair comparisons are required. b) When the first letter of *Surname* is invoked as a blocking variable, only record pairs having the same first letter of their *Surname* are compared to one another. Other record pairs remain unexamined and are shown in gray. Under this blocking scheme, only 5 record pair comparisons are considered and subsequently classified.

matches, shown in green, 16 record pairs must be considered. In Figure 2.2(b), a blocking strategy is utilized, such that only record pairs for which the first letter of the *Surname* agrees are compared to one another. This removes from consideration 11 record pairs – shown in gray – and reduces the number of record pairs that must be compared to 5.

Blocking can also be viewed as a partitioning problem, as demonstrated in Figure 2.3. Similar records are grouped into the same partition, and only record pairs within the same partition are compared to one another, thus reducing the number of record pair comparisons required.

While blocking has great potential for reducing the number of record pair comparisons required, it may also negatively affect linkage accuracy if record pairs that are *True Matches* are removed from consideration during the blocking step. Therefore, care must be taken to balance reducing computational complexity without reducing accuracy.

A blocking method for use in PRL is described in Chapter 5.

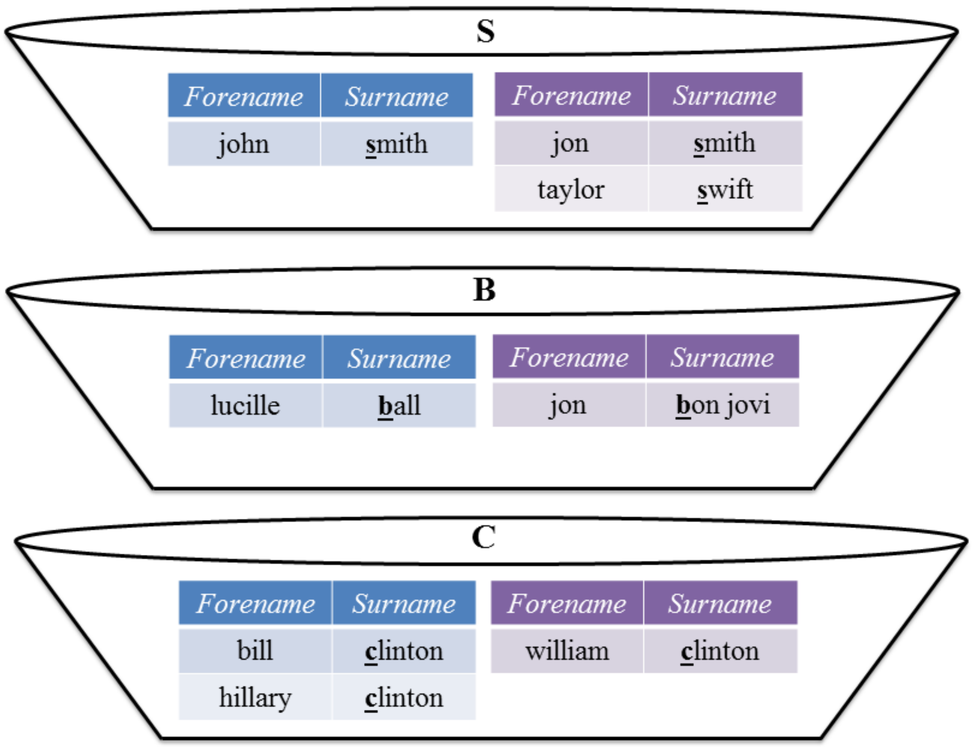
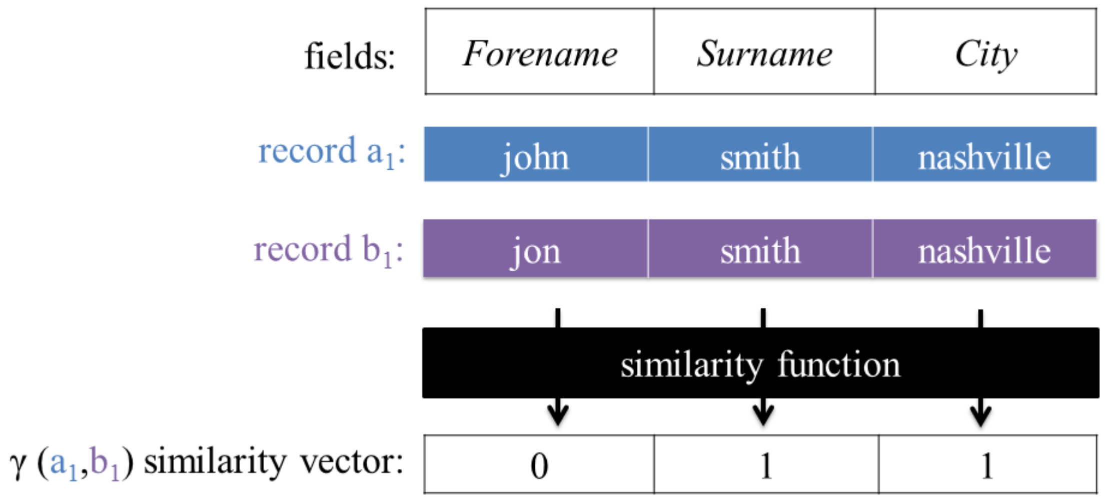


Figure 2.3: Blocking can be viewed as a partitioning problem. In this example, the first letter of *Surname* is invoked to partition the records.

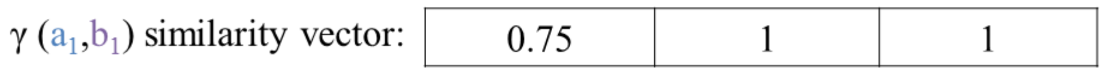
2.1.5 Field Comparison

Records are composed of attributes called fields (*e.g.*, *Surname*, *Forename*, and *City*). In the field comparison step of record linkage, a similarity function is applied to measure the similarity of two field values, as shown in Figure 2.4. The inputs to the field comparison step are the field values for two records. The output is a vector, with an entry corresponding to each field, representing the similarity of the field values of the record pair. This vector is often called the γ vector. Depending upon the similarity function used, similarities may be binary (*i.e.*, a value drawn from the set [0,1]) or continuous (*i.e.*, a real value drawn from the range [0,...,1]) to represent an approximate measure of similarity. Examples of binary and approximate similarity are shown in Figures 2.4(a) and 2.4(b), respectively.

A key component of PRL is private field comparison – the process of determining the similarity of two field values without knowledge of the plaintext values. Plaintext values are those that are legible to the human eye. Encoded field values are referred to as



(a) Binary field comparison



(b) Approximate field comparison

Figure 2.4: The field comparison step of record linkage. The field values of a given record pair are input to a similarity function, and the γ similarity vector corresponding to the record pair is output.

ciphertexts and are illegible to the human eye. Several techniques for performing private field comparison are explored in Chapters 3 and 4.

2.1.6 Field Weighting

In this step, fields are weighted according to their discriminatory power in resolving identity. For example, *Surname* has greater discriminatory power than *Gender* and should be weighted accordingly when determining record pair similarity. Fellegi and Sunter proposed an algorithm for determining field weights that is commonly used in record linkage applications today [46]. This algorithm is described below.

The Fellegi-Sunter Algorithm

The Fellegi-Sunter (FS) field weighting algorithm uses a statistical approach based on estimation of the probability that a field agrees amongst record pairs that are *True Matches*, the m value (Equation 2.1), and the probability that a field randomly agrees amongst record pairs that are *True Non-matches*, the u value (Equation 2.2).

$$m[i] = P\left(a_x[i] == b_y[i] \mid (a,b) \in M\right) \ \forall \ a_x \in A, \ b_y \in B \tag{2.1}$$

$$u[i] = P\left(a_x[i] == b_y[i] \mid (a,b) \in U\right) \ \forall \ a_x \in A, \ b_y \in B \tag{2.2}$$

where $m[i]$ is the m value for the i^{th} field, $u[i]$ is the u value for the i^{th} field, a_x is a record in dataset A, b_y is a record in dataset B, and "==" denotes field agreement.

An example of m and u values for sample fields are shown in Figure 2.5(a). In this example, *Forename* agrees amongst 80% of *True Matches* and agrees by chance amongst 0.035% of *True Non-matches.*

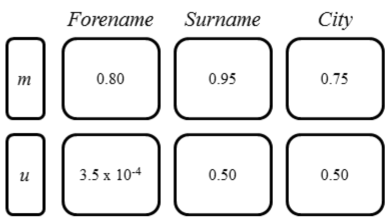
These calculations unfortunately require knowledge of the true match status of record pairs. Sometimes a subset of record for which the true match status is known can be used to calculate these probabilities. In the absence of such knowledge, an Expectation Maximization (EM) algorithm can be used to estimate the conditional probabilities associated with each field [49, 66, 108, 118].

The conditional probabilities are then used to calculate agreement weights, $w_a[i]$, and disagreement weights, $w_d[i]$, for each field $i = 1,\ldots,f$:

$$w_a[i] = \log\left(\frac{m[i]}{u[i]}\right) \tag{2.3}$$

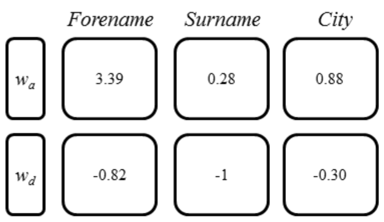
$$w_d[i] = \log\left(\frac{1-m[i]}{1-u[i]}\right) \tag{2.4}$$

Agreement and disagreement weights for sample fields are shown in Figure 2.5(b). To summarize, the conditional probabilities of field agreement given match status are calculated (or estimated) for each field. These conditional probabilities are then applied to calculate agreement and disagreement weights for each field. For an example, see Figure 2.5.



	Forename	Surname	City
m	0.80	0.95	0.75
u	3.5×10^{-4}	0.50	0.50

(a) Conditional probabilities.



	Forename	Surname	City
w_a	3.39	0.28	0.88
w_d	-0.82	-1	-0.30

(b) Agreement and disagreement weights.

Figure 2.5: The FS algorithm for field weighting. Sample conditional probabilities, and the associated agreement and disagreement weights, are shown.

2.1.7 Record Pair Comparison

In the record pair comparison step, the similarities of the fields in a record pair are consolidated into a single measure of record pair similarity, as demonstrated in Figure 2.6.

One method for record pair comparison is to apply the agreement and disagreement weights associated with each field to calculate a similarity score for the record pair. The similarity score is calculated for each record pair according to Equation 2.5.

$$\textit{similarity score}(a_x, b_y) = \sum_{i=1}^{f} w_a[i]^{\gamma_{(a_x,b_y)}[i]} \times w_d[i]^{1-\gamma_{(a_x,b_y)}[i]} \tag{2.5}$$

Therefore, if field i agrees, the agreement weight associated with field i, $w_a[i]$, will be added to the total score for the record pair; if field i disagrees, the disagreement weight associated with field i, $w_d[i]$, will be added to the total score for the record pair, as shown in Figure 2.6.

At this point, we would like highlight the terminology used when discussing field values as compared to record pairs. We use the terms "agree" and "disagree" when referring to individual field values within a record pair, and the terms "*Predicted Match*" and "*Predicted Non-match*" when discussing the classification status of the entire record pair. Therefore, it is possible for some field values to disagree within a record pair classified as a *Predicted Match*, and similarly for some field values to agree within a record pair classified as a *Predicted Non-match*.

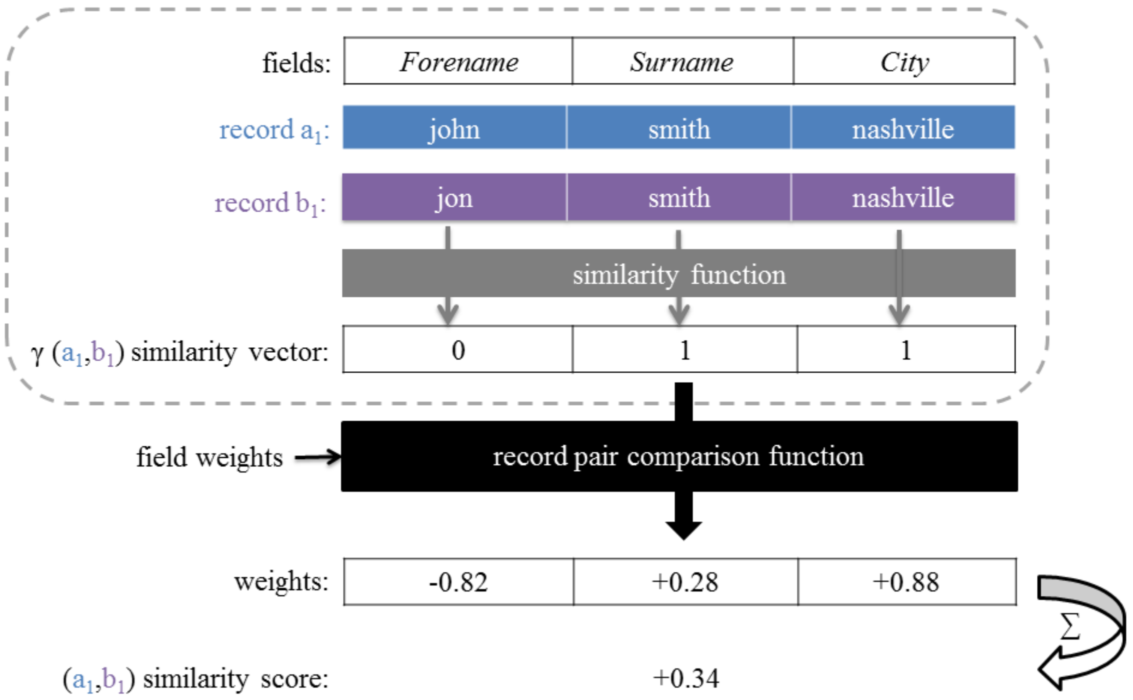


Figure 2.6: The record pair comparison step of record linkage. In this step, the γ field similarity vector, calculated in the field comparison step shown in the gray dotted box, and the field weights are input to a record pair comparison function. The output is a single measure of the similarity of the record pair.

2.1.8 Record Pair Classification

In the record pair classification step, a classifying boundary is drawn between record pairs to form the classes *Predicted Matches* and *Predicted Non-matches*. In traditional record linkage, a third intermediate class may by included, in which record pairs are set aside for clerical review to determine the true match status. However, in the context of PRL, clerical review clearly compromises privacy and thus this third, intermediate class is excluded.

2.2 Protocol

Within this dissertation, we consider the linkage of records from two data holders, to whom we refer as Alice and Bob. The following PRL protocol is then followed.

1. First, Alice and Bob exchange the information required to encode their records.
2. Next, Alice and Bob locally encode their records and send these encodings to a third party, Charlie.
3. Finally, Charlie executes the linkage protocol.

We adopt the third party model based on the observation that such parties exist in real world privacy-preserving data sharing environments (*e.g.*, [94]). In addition, in many cases, the involvement of third parties provides for more efficient protocols [34].

2.3 Implementation Details

Several computers were used for the experiments described in this research. Details of these computational resources are summarized in Table 2.1. Several machines are required, due to the computational complexity and scale of the experiments. Therefore, while running times reported throughout this dissertation are not all comparable, each set of experiments intended to be comparable are run on the same machine. Throughout the remainder of this

Computer ID	System memory	Processor	Operating system
δ	4 GB	2.5 GHz, quad-core	Ubuntu
ϵ	16 GB	2 GHz, octa-core	Ubuntu server
ζ	4 GB	2 GHz dual-core	Microsoft Windows Server 2003
θ	16 GB	3.4 GHz, octa-core	Ubuntu
ι	26 GB	2.2 GHz, 48 multi-core	CentOS

Table 2.1: Computational resources used for experiments.

dissertation, when experiments are described, the set of computational resources utilized for the experiments is noted.

Unless otherwise indicated, all methods are implemented in Perl, a language known for its string handling capabilities. The running time is measured using the Time::HiRes package. Datasets are stored as flat text files unless otherwise noted.

2.4 Description of Datasets

For the purposes of empirical evaluation, two datasets are studied in this work. The first, described in Section 2.4.1, is a large, publicly available dataset based on voter registration records. The second, described in Section 2.4.2, is based on patient information, and is therefore not publicly available.

2.4.1 North Carolina Voter Registration Records

To provide others with the opportunity to evaluate our research on the same dataset, the publicly-available North Carolina voter registration (NCVR) records [5] are used throughout this research. The fields contained in these records are shown in Table 2.2. This dataset contains 6,190,504 individual records.

Datasets for record linkage research are notoriously difficult to obtain for several reasons. The first step in creating a dataset for record linkage research is to obtain records from multiple data holders pertaining to a single population. However, regulations on data sharing can make this difficult [31]. The second step in creating a dataset for record linkage research is to manually curate the records to determine the true match status of record pairs. This can be laborious and unwieldy because the number of record pairs requiring classification is quadratic with respect to the number of records in the files.

To sidestep this difficulty, we implement a "data corrupter" based on the research of Christen and Pudjijono [31]. The corrupter introduces optical character recognition errors (*e.g.*, *S* swapped for 8), phonetic errors (*e.g.*, *ph* swapped for *f*), and typographic errors, such as insertions, deletions, transpositions, and substitutions. The corrupter introduces errors at rates consistent with the error rates seen in real datasets [31]. Phonetic errors are introduced with probability 3%, optical character recognition errors with probability 1%, insertions with probability 5%, deletions with probability 15%, substitutions with probability 35%, and transpositions with probability 5%.

In addition to these character-level errors, we extend the data corrupter to introduce token-level errors. These errors are introduced at frequencies estimated to be encountered in real datasets. *Nicknames* are substituted for *Forenames* (and vice-versa) with probability 15%, *Street address*es are changed to represent an individual moving, with probability 10%, *Surname*s are changed with probability 10% for females (due to the common practice of taking the husband's surname), 1% for males, and *Surname*s are hyphenated with probability 1%. The *Nickname*s are based on the Massmind Nicknames Database [4], the *Surname*s are based on the 2000 U.S. Census names dataset [111], and the *Street address*es are selected from a subset of the records not used in the record linkage datasets. Figure 2.7 provides an example of two records and their corrupted counterparts.

Therefore, for the NCVR dataset, given a dataset A, a second dataset B is generated by corrupting each record in A. The record linkage experiments can then be tested using record files A and B. While a "true" record linkage dataset – where the datasets are constructed independently – would be ideal, datasets created in the manner described are useful for benchmarking and testing new ideas.

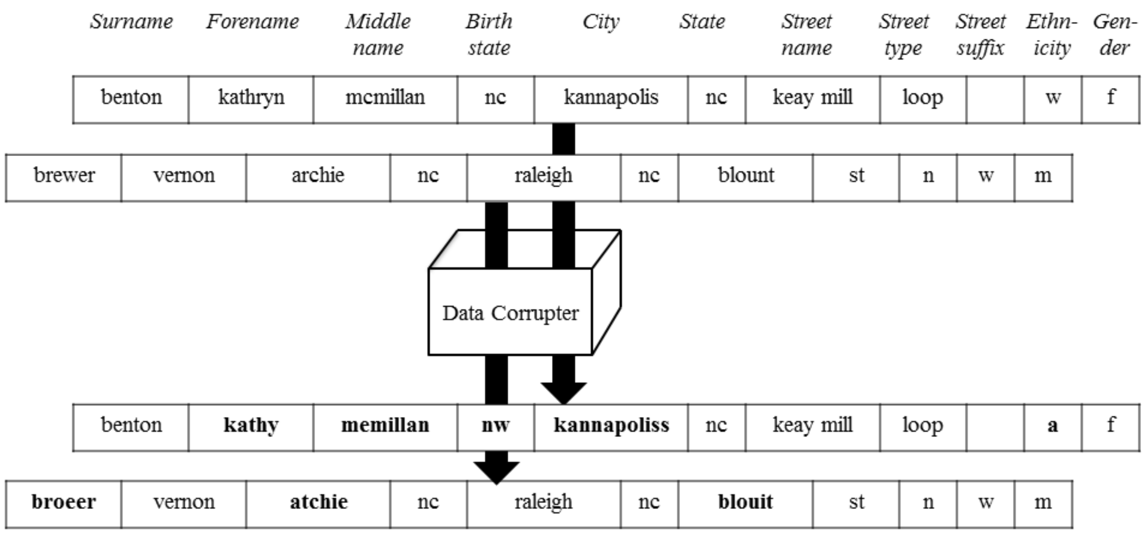

Surname
Forename
Middle name
Birth state
City
State
Street name
Street type
Street suffix
Ethn- icity
Gen- der
benton | kathryn | mcmillan | nc | kannapolis | nc | keay mill | loop | | w | f
brewer | vernon | archie | nc | raleigh | nc | blount | st | n | w | m
Data Corrupter
benton | **kathy** | **memillan** | **nw** | **kannapoliss** | nc | keay mill | loop | | **a** | f
broeer | vernon | **atchie** | nc | raleigh | nc | **blouit** | st | n | w | m

Figure 2.7: Sample records and their corrupted counterparts. Fields affected by corruption are highlighted in bold.

Further details about the NCVR dataset are provided in Table 2.2. In general, there are more unique values in the corrupt dataset (dataset B) because random errors have been introduced. Additionally, there are fewer missing values in the corrupt dataset because substitutions are sometimes introduced when the original value was missing.

2.4.2 Vanderbilt Patient Records

In addition to the NCVR dataset, we evaluate our work using a real dataset composed of identifiers and demographics from patient records. The records were extracted from StarChart, an electronic medical record system in use at Vanderbilt University Hospital. Vanderbilt University Hospital is an academic tertiary care facility with over 500 adult beds and 34,000 admissions annually.

The records were manually reviewed by the medical records department using reports from clinicians, chart inspection, and consultation of demographics records to construct a "gold standard" true match status. The review was accomplished in the context of clinical operations and was completed before our study began. The Vanderbilt dataset contains 36,658 patient records. The fields in this dataset and other details are provided in

	# Unique values		# Missing values	
	Dataset A	Dataset B	Dataset A	Dataset B
Surname	274,790	1,571,919	0	0
Forename	187,742	912,132	39	22
Middle name	263,689	986,388	409,843	250,309
Birth state	67	14,802	980,455	599,192
City	793	596,650	342	214
State	19	5,139	69	40
Street name	94,343	2,248,029	345	219
Street type	190	25,716	202,248	123,618
Street suffix	16	1,136	5,767,388	3,526,782
Ethnicity	11	1,282	66	38
Gender	7	1,030	1	0

Table 2.2: Details of the North Carolina Voter Registration dataset

Table 2.3.

As Table 2.3 indicates, many records in the Vanderbilt dataset contain missing field values. We chose not to include these incomplete records in many of our experiments because evaluating our methods without the complicating factor of missing data allows for a straightforward evaluation of each method. Once our methods have been evaluated in the straightforward setting, they can be modified to account for more complex scenarios, such as when field values are missing. Properly handling missing field values is imperative for the success of a PRL application however. Therefore in Chapter 6, we use the full Vanderbilt patient data to evaluate the complete PRL framework presented, and we evaluate different ways to handle missing data.

	# Unique values		# Missing values	
	Dataset A	Dataset B	Dataset A	Dataset B
SSN	25,552	23,069	11,084	13,571
Surname	12,479	14,335	13	37
Forename	7,878	7,778	3	10
Middle name	2,549	1,333	16,924	24,129
DOB	21,015	19,915	7	10
Gender	3	3	548	3,529
Ethnicity	7	7	19,465	26,166
Street address	27,534	23,972	8,481	11,903
City	988	861	8,261	11,047
State	43	44	8,265	11,235
Zip code	1,149	1,003	8,307	11,267
Telephone	30,351	25,780	5,385	10,203

Table 2.3: Details of the Vanderbilt dataset

CHAPTER III

EVALUATION OF PRIVATE STRING COMPARATORS

PRL requires the application of field comparators that do not require knowledge of the plaintext field values. A number of private string comparators have been proposed, but little research has compared them in the context of a real record linkage application. This chapter describes a principled and comprehensive evaluation of six private string comparators in terms of three key properties: 1) correctness of record linkage predictions, 2) computational complexity, and 3) security. A publicly-available dataset, derived from the North Carolina voter registration dataset, is used to evaluate the tradeoffs between the aforementioned properties. A notable result is that private string comparators that tokenize, encode, and compare strings yield highly accurate record linkage results. However, as a tradeoff, we observe that such private string comparators are less secure than those that map and compare strings in a reduced dimensional space.

3.1 Introduction

As discussed in Chapter 1, the sharing of plaintext data for record linkage is not always possible due to societal concerns and regulatory constraints. As a result, the process of PRL has been established as a variant of the task in which the data owners can perform record linkage without revealing the identities of individuals to whom records correspond [34].

The focus of this chapter is the field comparison step of record linkage, as discussed in Section 2.1.5. Specifically we examine methods for private field comparison that allow one to determine the similarity of two field values without knowledge of the plaintext field values.

A number of private string comparison techniques have been proposed in the litera-

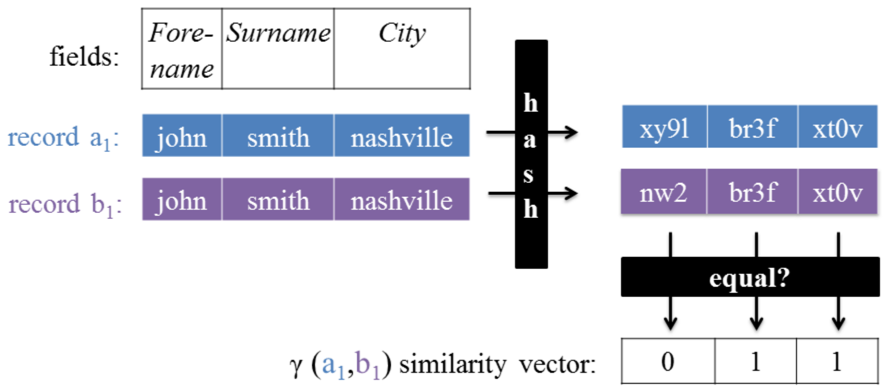


Figure 3.1: Private field comparison based on hashed field values.

ture, begging the question, "Which is the best method"? It should first be recognized that PSCs are generally designed to compare encoded versions of an individual's identifying values. Unfortunately, many of these approaches function only when field values are recorded consistently and without error across disparate data sources [8, 18, 45]. This neglects the fact that variation (*e.g.*, *Nickname* used in place of *Forename*) or typographical error can corrupt personal identifiers [55]. In such cases, the application of models that only detect equivalence can result in subpar record linkage results. For example, the hashed value of "Jon" appears equally distant from the hashed values of "John" and "Jiminy" whereas, in fact, "Jon" is far more similar to "John" than to "Jiminy"[*]. See Figure 3.1 for an example of private field comparison based upon a hashed version of field values. Hashing is not a similarity-preserving encoding, so the system loses the ability to determine the approximate similarity of hashed values. The only measure of similarity, given this type of encoding, is binary, meaning that it can be determined only if the field values agree completely (*i.e.*, a similarity of 1) or do not agree completely (*i.e.*, a similarity of 0).

In recognition of this limitation, various approximate comparators have been developed to measure string similarity, some of which have been translated into PSCs [18, 32, 45, 103]. While there is clear evidence (*e.g.*, Porter *et al.* [93] and Tromp *et al.* [110]) that accounting for string similarity can improve record linkage results, it is unclear which PSC

[*]A cryptographic hash function is a deterministic encoding function that converts a plaintext string into a fixed-length encoding.

is best for PRL applications.

We recognize that several reviews of PSCs have been conducted [109, 115], but they lack either the breadth (*i.e.*, range of methods considered) or the depth (*i.e.*, a quantitative evaluation of the correctness, security, and computational complexity) necessary to compare approaches on a common scale. For instance, Elmagarmid and colleagues [44] presented a comprehensive survey of record linkage techniques, but did not consider the privacy issues and the related PRL techniques. Meanwhile, Trepetin [109] performed a brief review of PSCs in association with the introduction of a new PSC, but the review focused more on a qualitative discussion of computational complexity, rather than record linkage correctness and security of the various approaches. Another paper [53] provides an overview of PRL, current approaches, and open research questions, but again does not provide a formal, quantified analysis of existing approaches. In another study, Verykios and colleagues [115] performed a principled review of PSCs, but focused on string comparison more generally, rather than string comparison within the context of record linkage. Moreover, the latter study also neglected a formal security analysis of the PSCs.

3.1.1 Contributions

Given the current state of affairs, in order for data holders to perform PRL, they must sift through a variety of approaches that have not been directly compared to one another. Therefore, there is a need for a comprehensive evaluation of existing PSCs that places the comparators on a level playing field. The research described in this chapter makes the following contributions:

1. **Taxonomy:** We perform a comprehensive literature review of PSCs. In doing so, we derive a taxonomy of existing approaches, which classifies PSCs into distinct classes of techniques. Classifying the approaches allows for analysis of features that are common to each class of methods.

2. **Framework:** We introduce a PRL framework suitable for PSCs that measure approx-

imate field similarity, rather than field equivalence alone. Notably, this is an aspect of the broader PRL problem which has been neglected.

3. **Common Criteria:** We define several quantifiable measures to compare PSCs on common scales. These measures enable data holders to investigate the tradeoffs between PSCs on three critical axes, namely 1) correctness in record linkage, 2) computational complexity as measured by running time, and 3) security.

4. **Evaluation:** We systematically investigate the relationship between six classes of PSCs using a publicly available dataset. The fields in the dataset consist of personal identifiers and demographics that have been proposed as key attributes for record linkage (*e.g.*, *Forename* and *Street address*).

3.1.2 Chapter Outline

The remainder of this chapter is organized as follows. In Section 3.2, we describe related work and introduce a taxonomy of PSCs. In Section 3.3, we present the specific record linkage methodologies used in this work and the details of the dataset utilized in this evaluation. Additionally, Section 3.3 details the parameters used for each comparator in addition to the evaluation metrics. Then in Section 3.4, the correctness, computational complexity, and security metrics are reported for each comparator. Section 3.5 provides a discussion of factors affecting the performance of each comparator, limitations, and future work. Finally, we conclude and summarize this chapter in Section 3.6.

3.2 Related Work

In this section, we introduce a taxonomy for private field comparators. This enables us to generalize the strengths and weaknesses of each class of methods.

A review of the literature surrounding private field comparators revealed the six broad categories shown in Table 3.1. The PSCs empirically evaluated in this chapter are denoted in italics. Several noteworthy privacy-preserving field comparators, more suitable for numerical fields [61, 71, 95], such as *Age*, are considered, but are excluded from this study because our goal is to evaluate string, rather than numerical, comparators. We reiterate that private numerical comparators are important methods that may be necessary for a comprehensive PRL application; however, they are not the focus of the work described in this chapter.

Equivalence Testing	*Exact Agreement* [8, 18, 45]
***n*-gram Methods**	Number of Common Bigrams [32]
	Bloom Filter [101]
	Trigrams [60]
Reference Space Embedding	Public Reference Space Embedding [92]
	Private Reference Space *Embedding* [99]
	Dissimilarity Matrix [61]
Teamwork	*Edit Similarity* [12]
	Secret Sharing [71]
	Euclidean Distance [95]
Phonetic Filtering	*Phonetic Filter* [70]
Guess & Encode Errors	Random Introduction of Errors [103]
	Systematic Introduction of Errors [40]

Table 3.1: Taxonomy of private field comparators.

3.2.1 Exact Agreement Methods

As alluded to, many PSC protocols use variations of an “encode-and-compare” model in which identifiers are encoded and compared for equality [7, 8, 18, 45, 94]. When a plaintext string is encoded, it is transformed into a form known as a ciphertext. Comparators based on equivalence testing do not take into account the notion of similarity between the strings. This means that plaintexts very similar to one another generally result in ciphertexts that are very dissimilar to one another [104]. Our hypothesis is that leveraging similarity will improve record linkage results. To gauge the degree of improvement achieved through the use of approximate string comparators, we will use the *Exact Agreement* comparator as a baseline.

3.2.2 n-gram Methods

In the n-gram class of PSCs, strings are tokenized into the n-grams that compose the string, where n is an integer that can range from 1 to the length of the string. n-grams are commonly padded on both ends with $n - 1$ blank spaces to properly account for the first and last letters. For example, the 2-grams, also called bigrams, associated with the name "John" are " J", "Jo", "oh", "hn", and "n ". If padding were not used, all letters, except for the first and the last, would be included in two bigrams, and the first and last letters would only be included in a single bigram. In order to ensure that the first and last letters are included in the same number of bigrams as all other characters, the string is padded with a blank space. This means that the first and last letters are now included in two bigrams, in the same manner as the other letters.

Number of common bigrams

Churches and Christen proposed an approximate PSC based on determining the number of bigrams shared in common between strings [32]. However, previous evaluations have already demonstrated this comparator did not perform well in record linkage and is too computationally intense to be practical [44, 109]. Therefore, we do not evaluate this comparator in this study.

Trigrams

The comparator proposed by Hylton *et al.* [60] uses the set of encoded 3-grams, or trigrams, associated with the string. When two strings are compared, a difference vector D is calculated (see Table 3.2) where each cell represents the difference in the number of times a trigram appears in each string. To determine the similarity of two strings, a threshold T is used[†]:

$$T = 2.486 + 0.025C \tag{3.1}$$

[†]This threshold was experimentally determined by Hylton *et al.* [60] and was shown to work well in practice.

where C is the number of unique trigrams contained among the set of trigrams belonging to both strings. If the magnitude of the difference vector D is less than the threshold T, the strings are assigned a similarity score of 1 (*i.e.*, the strings are completely similar); otherwise, the strings are assigned a similarity score of 0 (*i.e.*, the strings are completely dissimilar).

unique trigrams:	h(“ j”)	h(” jo”)	h(“joh”)	h(“jon”)	h(“ohn”)	h(“hn ”)	h(“ on”)	h(“n ”)
john:	1	1	1	0	1	1	0	1
jon:	1	1	0	1	0	0	1	1
D:	0	0	1	1	1	1	1	0

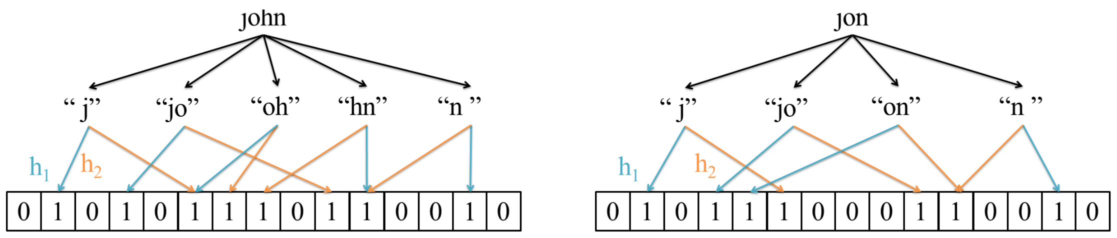
Table 3.2: The *Trigrams* comparator, where h represents a hash function. The vector D contains the difference in the count of trigrams found in each bag. $|D| = \sqrt{5} = 2.24$. $T = 2.486 + 0.025\mathrm{C} = 2.486 + (0.025 \times 8) = 2.7$. Since $|D| < T$, the similarity is determined to be 1.

Bloom Filter Encoding

A Bloom filter is a data structure commonly used to determine set membership or to compare sets of stored values [24, 25, 65]. It is represented as a bit array where all bits are initialized to 0. To store a value in a Bloom filter, a hash function is applied to the value. The hash value corresponds to an index in the Bloom filter, and the bit value at that index is set to 1. This is repeated k times with k different hash functions.

As multiple values can be stored in the same Bloom filter, a bit set to 1 by the previous item can be set again by the newly added item. In this case, the value of the bit remains at 1.

A user can then issue a query to determine if a given value is stored in a Bloom filter. To determine if a given value has been encoded in a Bloom filter, the query issuer applies the k hash functions to the query value to generate the set of bits corresponding to the value. If all k bits corresponding to the query value are set to 1, the value may be stored in the Bloom filter. It may be the case that all k bits corresponding to the value were incidentally set by other values stored in the Bloom filter, and that the value in question was actually not stored in the Bloom filter. For this reason, false positives are possible. If



(a) The Bloom filter encoding of the name "john". (b) The Bloom filter encoding of the name "jon".

Figure 3.2: The *Bloom filter* comparator. In this example $k = 15$, $m = 2$, and h_1, h_2 are the hash functions applied. The Dice coefficient (Equation 3.2) is used to determine the similarity of the Bloom filters. The number of intersecting bits in the Bloom filters is 6, the number of bits set in the Bloom filter shown in part a is 8, and the number of bits set in the Bloom filter shown in part b is 7, yielding a similarity of 0.8 according to Equation 3.2.

any of the k bits corresponding to the given value is not set to 1, the query issuer can be assured that the value is not stored in the Bloom filter. This means that false negatives are not possible.

Schnell *et al.* introduce an application for Bloom filters [101] in PRL in which each field value is encoded in a Bloom filter. In this application, an encoding for each string is generated by hashing the bigrams associated with the string into a Bloom filter using k different hash functions. When Alice and Bob compare their strings, they agree upon the number of bits, m, that compose the Bloom filter, the value n to use in n-gram generation, the number of hash functions, k, to apply, and which hash functions to utilize. Alice then encodes her string by hashing its corresponding bigrams into a new Bloom filter using all of the k agreed upon hash functions. Bob encodes his string similarly. To determine the approximate similarity of their strings, the corresponding Bloom filters are compared using a set-based similarity measure, such as the Dice coefficient, d:

$$d(\alpha, \beta) = 2\left(\frac{|\alpha \cap \beta|}{|\alpha| + |\beta|}\right) \tag{3.2}$$

where α and β are the Bloom filters containing the hashed bigrams of Alice's and Bob's encoded strings, respectively.

3.2.3 Embedding Methods

The comparators in the Embedding class use a set of reference strings to define an embedding space. Alice and Bob embed a string in this space by calculating the distance (determined by some distance function[‡]) between their string and the reference sets, resulting in a vector of distances. To compare the strings, Alice and Bob compare the distance vectors associated with each of their strings. The idea behind this approach is that if the strings are similar, they will also be of similar distance to the reference strings in the embedding space.

Embedding with a public reference space

The embedding approach proposed by Pang *et al.* [92] suggests using a publicly available reference table to define the embedding space.

Embedding with a private reference space

The approach introduced by Inan *et al.* [61] recommends that Alice and Bob generate the reference space, which is kept secret from Charlie. We select this comparator for evaluation because a privately held reference space provides greater security than a publicly available reference space. For simplicity, this comparator will henceforth be referred to as *Embedding.*

The *Embedding* approach [61] is based on the use of the Sparse Map variant of Lipschitz embeddings. The general idea is that the embedding space is defined by several reference sets, each of which contain randomly generated reference strings. When a string is embedded in this space, its distance to each of those reference sets is measured. The distance from a string s to a reference set is the minimal distance between s and each string composing the reference set. The embedding of a string is therefore a vector of distances, where each element j denotes the distance from the string to reference set j. To compare strings, the Euclidean distance of their embeddings is calculated. A greedy resampling heuristic is used to limit the computational complexity by reducing the dimensionality of the embeddings such that, rather than using all possible reference sets, only a subset of the

[‡]While any distance function can be used, selection of an appropriate, informative function is important for achieving accurate record linkage results.

most information-rich reference sets are used. A proper explanation is beyond the scope of this dissertation, but we refer the reader to Inan *et al.* [61] for the *Embedding* comparator and Hjaltason *et al.* [58] for Sparse Map.

3.2.4 Teamwork Methods

The comparators in the teamwork class are so-called because they require that Alice and Bob interactively cooperate throughout the protocol to compare their strings.

Edit Similarity

The Levenshtein edit distance between two strings is the minimal number of insertions, deletions, and substitutions required to transform one string into the other [79]. A dynamic programming algorithm is often used to calculate edit distance, such that the minimal number of edits between the strings is iteratively calculated. For example, if the edit distance between strings of length $|a_x[i]|$ and $|b_y[i]|$ is calculated, a matrix of size $|a_x[i]|$ times $|b_y[i]|$ is set up. At the completion of the algorithm, the matrix cell $M[|a_x[i]|][|b_x[i]|]$ represents the minimal edit distance between the two strings. Further details can be found in the work by Marzal *et al.* [85].

Atallah and colleagues present a comparator for securely calculating edit distance without revealing either string [12]. In this protocol, the standard dynamic programming algorithm for edit distance is applied with the exception that the matrix and its values are split between Alice and Bob. Alice and Bob fill in the cells of their respective matrices using a protocol that allows them to determine the optimal method for calculating matrix values, without revealing any information to one another. At the completion of the protocol, Alice and Bob encode and send the value in the final cell of the matrix to the third party Charlie. Charlie sums the values he received from Alice and Bob to determine the final edit distance.

While the original method presented by Atallah and colleagues is edit distance, a more informative variant of this metric, edit similarity, was selected for evaluation in this manuscript. The difference between edit distance and edit similarity is discussed in Section

$$\text{john} \rightarrow \text{soundex(john)} = \text{J500} \rightarrow \text{hash(J500)}$$
(a) Encoding "john".

$$\text{jon} \rightarrow \text{soundex(jon)} = \text{J500} \rightarrow \text{hash(J500)}$$
(b) Encoding "jon".

$$\text{hash(J500)} == \text{hash(J500)} \rightarrow \text{similarity} = 1$$
(c) Comparing the encodings of "john" and "jon".

Figure 3.3: The *Phonetic Filter* comparator. The Soundex code for each string is hashed and checked for equality. The hashed Soundex codes are equivalent ("==" denotes equality), so the similarity of the strings is 1.

3.3.2.

3.2.5 Phonetic Filtering Methods

Phonetic Filter

Phonetic filters are used to generate a phoneme-based representation of a string. This representation can overcome various errors, such as typos and the use of nicknames. For example, "john" and "jon" are mapped to the same phonetic encoding (see Figure 3.3). Multiple phonetic encoding strategies, such as Soundex [70], Metaphone [23] , and NYSIIS, [51] have been used in record linkage applications. Karakasidis and Verykios suggest use of the Soundex phonetic filter to transform a string into its phonetic representation and then encode the phonetic representation [70]. If the encoded phonetic representations agree, the strings are assigned a similarity value of 1; otherwise, they are assigned a similarity value of 0.

3.2.6 Guess & Encode Errors Methods

When strings are encoded and compared, small differences in the plaintext strings can result in very different ciphertexts. To overcome the small differences that can be created through typographical, spelling, or other errors in strings, a class of of approaches have been proposed that attempt to preemptively generate errors that might be associated with a string. Rather than compare a single pair of strings, the sets of error-riddled relatives of the original strings are compared [40, 68, 103]. When Alice and Bob compare their strings, if any of the error-riddled relatives of Alice's string match any of the error-riddled relatives of Bob's string, the strings are declared a match. However, these comparators result in many false positives, weak matching precision, and require unreasonable amounts of storage [109].

3.3 Materials & Methods

The previous section presented a taxonomy of existing PSCs. However, string comparison is only one part of the larger record linkage process as described in Section 2.1. In this section, we provide the specific approaches used in each step of record linkage. Finally, the dataset and implementation details are provided.

3.3.1 Details of Record Linkage Process

As briefly described in Section 2.1 and depicted in Figure 2.1, record linkage involves multiple steps. In the field comparison step, each field in each record pair is compared, resulting in a vector, called γ, of size f, where f is the number of fields in each record. Each cell in the vector, $\gamma(a_x, b_y)[i]$, indicates the similarity of field i in record pair (a_x, b_y). In the record pair comparison step, the f-sized γ similarity vector for each record pair is converted into a single similarity score for the record pair. In the record pair classification step, the pairs are partitioned and classified into either the set M or U. The specific instantiations of these steps in this study are as follows:

- *Field comparison*: Six experimental string comparators (Section 3.3.2) are used for field comparison. Additionally, a plaintext string comparator shown to work well in record linkage [35, 50] is used as a control.

- *Record pair comparison*: A modification by Porter and Winkler [93] to the FS field weighting algorithm is used for record pair comparison. The details of this method are provided below.

- *Record pair classification*: We use the following approach to classify all record pairs into M and U based on their similarity scores. Each dataset A studied in this chapter contains 1,000 records and is linked to a dataset B that also contains 1,000 records. Each record in A must be compared to each record in B, which results in $1,000 \times 1,000 = 10^6$ record pairs that must be classified into the sets M and U. Each record in A has a record in B that refers to the same individual, so we know 1,000 of the record pairs, or 0.1%, are *True Matches.* Therefore, we classify the 1,000 record pairs having the highest scores as M and all other record pairs as U[§].

Throughout these experiments, we hold constant the methods for both record pair comparison and classification so that we can clearly focus on the field comparison step.

Modification to FS field weighting algorithm to provide for approximate field comparison

This section provides the details of Porter and Winkler's modification to the FS field weighting algorithm to provide for approximate field comparison. The original FS algorithm provides field weightings that can be applied during the record pair comparison step to assigned an overall similarity score to the record pair (see Sections 2.1.6 and 2.1.7 for further details). However, it is intended to work with only binary field comparison metrics. To incorporate approximate field comparators, that detect similarity in addition to simply agreement or disagreement, the FS algorithm must be modified.

Porter and Winkler introduced a modification that provides for approximate field comparators, which yield a continuous score in the range [0,. . . ,1] [93]. Under the modification, the conditional probabilities, agreement weights, and disagreement weights are calculated in the manner described in Section 2.1.6. However, the agreement weights now lay out a scale onto which the field comparison scores are mapped. A field similarity of 0 maps to the value of the disagreement weight, and a field similarity of 1 maps to the value

[§]This method does not enforce a one-to-one mapping of records. In practice, this may be enforced through an exhaustive linear sum assignment procedure or a greedy matching heuristic. Further details can be found in the work by Lenz *et al.* [78]

of the agreement weight. Intermediate field similarities map to intermediate score weights. For example, if the field similarity score is 0.75, then the score at the 75^{th} percentile on the scale laid out by the agreement and disagreement weights is selected. For an example, see Figure 3.4. More formally, the score calculated according to this method is given as follows:

$$score(a_x, b_y) = \sum_{i=1}^{f}((w_a[i] - w_d[i]) \times \gamma_{(a_x,b_y)}[i]) + w_d[i] \tag{3.3}$$

3.3.2 Details of Methods Evaluated

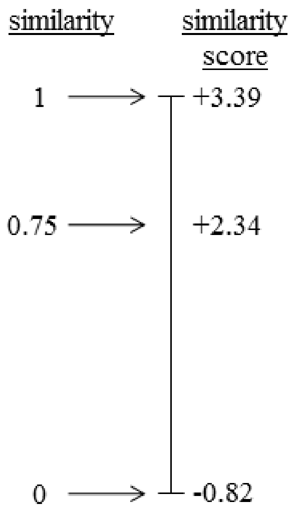
We select a subset of the aforementioned PSCs for evaluation in this chapter. PSCs that have performed poorly in previous evaluations were not considered [32, 40, 103]. To comprehensively evaluate the range of techniques, a comparator was selected from each of the categories described above, with the exception of the Guess & Encode methods, which have already been shown to perform poorly [109]. The PSCs selected, and the parameters invoked, are discussed in the remainder of this section. In all cases, we attempt to use the parameters suggested in the original publications. Where this is not possible, or parameters were not suggested, this is explicitly stated.

Exact Agreement

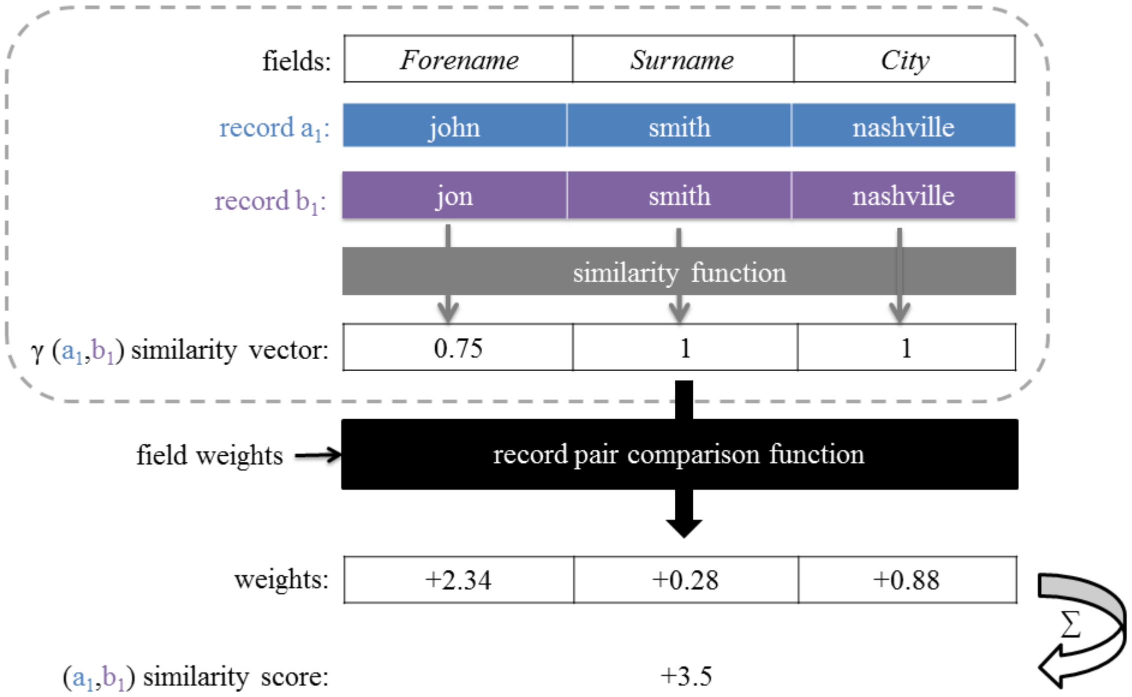
For the *Exact Agreement* comparator, each identifier is hashed (by the widely-used SHA-1 hash function) and compared. The output of the comparison is 1 if the hashed strings match exactly, and 0 otherwise. Each string is concatenated with a random string known as "salt" to prevent a dictionary attack, an exhaustive hashing of plaintexts to determine which encoded value corresponds to each plaintext value [100].

Bloom Filter Encoding

The *Bloom Filter* comparator is evaluated because it showed promising results in a preliminary evaluation we conducted [41]. As recommended in the original publication [101], we use a Bloom filter of length 1,000 bits and use 30 hash functions, all variations of SHA-1,



(a) Similarities are mapped to field weights.



(b) Record pair comparison using approximate measures of field similarity.

Figure 3.4: Porter-Winkler modification to the Fellegi-Sunter algorithm.

to hash each bigram into the Bloom filter. Strings are padded with spaces on both ends in bigram creation as described in Section 3.2.2.

Trigrams

The *Trigrams* [60] comparator was favorably reviewed [115] and is therefore selected for evaluation in this work. Each string is padded on both ends with two spaces in tokenization. SHA-1, in conjunction with “salt”, is applied to hash each trigram. A space-efficient implementation is used where each string is represented as a bag of trigrams and their associated counts. This is in contrast to representing each string as a very sparse vector of all possible trigrams, where the value for the cell associated with each trigram is the number of times the trigram occurs in a given string.

Embedding

The *Embedding* approach described by Inan *et al.* [61] is selected to represent the Reference Space Embedding class due to its ability to provide greater security. With respect to the parameters for this comparator, we attempt to follow the recommendations of the authors [61] as much as possible. As such, twenty strings were used in generating sixteen reference sets. For each field, the length of the reference strings is the average length of strings in the field. With respect to the greedy resampling heuristic, 10% of random pairs were sampled (without replacement) as recommended by Hjaltason *et al.* [58]. To determine the optimal number of coordinates in the final embeddings, we systematically test all values in the range [2, 16]. We found that using 9 coordinates produces the most accurate record linkage results in a small test set, and is therefore selected as the dimensionality of the final embeddings. Details can be found in Appendix A.

Euclidean distance is used to determine the similarity between embeddings. For each field, the Euclidean distance is normalized by the largest Euclidean distance seen to ensure the similarities are in the range [0,1].

Phonetic Filter

In the *Phonetic Filter* comparator, strings are transformed by the Soundex phonetic filter, encoded, and tested for equivalence. Salting is used to protect against dictionary attack. The Perl Text::Soundex package is used to encode field values.

Edit Similarity

The *Edit Similarity* approach [12] is selected for evaluation due to its rigorous treatment of privacy protection. We draw attention to the distinction between distance and similarity. Distance is in the range [0,∞], where lower distance indicates strings are more alike. Similarity, on the other hand, is in the range [0,1], where 1 indicates strings are completely alike and 0 indicates the strings are completely unalike. Distance can be converted to similarity as follows:

$$similarity\left(string_1, string_2\right) = \frac{distance\left(string_1, string_2\right)}{max\left(|string_1|, |string_2|\right)} \tag{3.4}$$

While the original comparator, as presented, returns the edit distance between two strings, we believe that edit similarity is a more informative measure because it incorporates string length. For example, an edit distance of three between the strings "Bob" and "Jan" is very different from an edit distance of three between the strings "Catherine" and "Katerina". The edit distance protocol described above can be modified to report edit similarity when Alice and Bob divide their final answers (*i.e.*, the last cell in their matrices) by the maximum string length. Note, this does not reveal any additional information as Alice and Bob already know the string lengths because they are required by the protocol.

3.3.3 Controls

To compare the PSCs to a control comparator, we use the *Jaro-Winkler* distance, a comparator was designed for assessing plaintext strings, that has been shown to work very well in record linkage [35, 50, 93]. Winkler introduced a modification to the Jaro distance that

$$d_j(john, jon) = \frac{1}{3}(\frac{3}{4} + \frac{3}{3} + \frac{3}{3}) = 0.9167$$
$$d_{jw}(john, jon) = 0.9167 + (2 \times 0.1 \times 0.0833) = 0.9$$

Figure 3.5: The *Jaro-Winkler* distance between the strings "john and "jon". The parameters used are $\chi = 3, \psi = 0, v = 0.1, l = 2$.

incorporated his observation that errors are more likely to occur at the end of a string, rather than the beginning [93]. The modified comparator therefore places greater emphasis on the characters at the beginning of the string in determining string similarity. The *Jaro-Winkler* string comparator is defined as:

$$d_{jw}(s_1, s_2) = d_j(s_1, s_2) + (lv(1 - d_j)) \tag{3.5}$$

where d_j is the Jaro distance for strings s_1 and s_2, l is the length of the prefix common to both strings (up to 4 characters), and v is a scaling factor. The default value for v is 0.1. The Jaro distance [66] is defined as:

$$d_j(s_1, s_2) = \frac{1}{3}\left(\frac{\chi}{|s_1|} + \frac{\chi}{|s_2|} + \frac{\chi - \psi}{\chi}\right) \tag{3.6}$$

where χ is the number of agreeing characters and ψ is the number of transpositions. A transposition is the swapping of two adjacent letters (for example, "jo*hn*" and "jo*nh*"). Two characters c_1 and c_2 are considered agreeing if they are no further than $\lfloor \frac{max(|c_1|,|c_2|)}{2} \rfloor - 1$ characters apart.

The *Jaro-Winkler* (JW) comparator is highly dependent on heuristics, but it has proven an excellent string comparator in record linkage by several evaluations [35, 50, 93]. Therefore, we use this comparator as the reference standard.

3.3.4 Evaluation Metrics

We evaluate each string comparator along three axes: 1) correctness in record linkage, 2) security (the extent to which the encoded values are protected in the linkage process), and 3) computational complexity as determined by the running time.

Correctness in Record Linkage

To evaluate correctness in record linkage, we examine the ability of each string comparator to accurately classify record pairs into the sets M and U. As mentioned earlier, only 1,000 of the possible record pairs are *True Matches.* The number of *True Non-matches* (999,000) dominates the number of *True Matches* to such an extent that some measures, such as specificity, are not very informative. Therefore, to focus on the 0.1% of record pairs that are *True Matches*, the True Positive (TP) rate¶ is examined for each comparator. The TP rate is defined as:

$$TP\ rate = \frac{TP}{TP + FP} \tag{3.7}$$

where TP is the number of true positives (*i.e.*, *Predicted Matches* that are *True Matches*) and FP is the number of false positives (*i.e.*, *Predicted Matches* that are *True Non-matches*). Therefore, the TP rate reports the proportion of *Predicted Matches* that are, in fact, *True Matches.* A comparator that perfectly classifies the record pairs has a TP rate of 1.

Computational Complexity

We use the running time to evaluate the computational complexity of each comparator. In this chapter, a record linkage method that requires more than one day to process the linkage of two record files, each containing 1,000 records, is considered computationally infeasible for real world use. We make this assumption because the datasets used in this work are relatively small compared to typical real world record linkage datasets. Additionally, the running time increases quadratically with the number of records. Therefore, comparators that take more than one day to run on these small datasets would take much longer to run on larger datasets.

¶The TP rate is also referred to as *Precision.*

Security

We next study the extent to which each comparator protects the security of the records. Recall that in the framework presented in Section 2.2, data owners Alice and Bob rely on a third party, Charlie, to perform the record linkage. Specifically, Alice and Bob first exchange the information required to encode their records (which varies for each comparator). This information can be viewed as a “key” which is then used to encode the records. Charlie receives the encoded records and performs record linkage without the key. To conduct a security analysis of each comparator, we make the following standard assumptions with respect to the knowledge and the behavior of all parties in the above protocol:

Assumption 3.1 *All parties strictly follow the protocol in data operations and communications. This means that Alice and Bob always encode the records using the shared encoding key. They will only send the encoded record to Charlie. None of the parties will maliciously change the content of the records or the linkage results.*

Assumption 3.2 *Charlie sees the encoded records, but not the plaintext records.*

Assumption 3.3 *Charlie does not know the encoding key.*

Assumption 3.4 *Charlie has knowledge of the distribution from which the records are drawn (in this case, the NCVR dataset).*

Under these assumptions, the security analysis of the PSCs focuses on *how much information Charlie can acquire from the encodings to infer the plaintext value.* Since each field is independently encoded in all comparators, we pick one field, *Surname*, to illustrate our metric for evaluating security. Here we use the *mutual information entropy*, a standard practice in security evaluations, to quantify the dependence of the plaintext and the ciphertext values [104].

Formally, let random variable N denote the original information for the selected field, which takes values $\omega_1 \ldots \omega_{|N|}$. Let $P(\omega_t)$ be the probability mass function of outcome ω_t. The entropy (H) of N is then defined as:

$$H(N) = \sum_{t=1}^{|N|} -P(\omega_t) \log P(\omega_t) \tag{3.8}$$

Similarly, we consider random variable O as the encoded form of this selected field, which takes values $o_1, \ldots, o_{|O|}$ and let $P(o)$ be its probability mass function. In our specific case, N refers to the plaintext *Surname*s present in the dataset and ω refers to each specific *Surname*. Thus, O refers to the encoded forms of each *Surname* associated with each comparator and o refers to each specific encoding. For example, in the *Bloom Filter* comparator, the encoded form is a string of 0s and 1s with length of 1,000 bits. The entropy (H) of O can be similarly defined as: $H(O) = \sum_{u=1}^{|O|} -P(o_u) \log P(o_u)$.

Let each pair of outcomes (ω_t, o_u) (*i.e.*, the encoded form of ω_t is o_u) occur with probability $P(\omega_t, o_u)$. The joint entropy of variables N and O is then defined as:

$$H(N, O) = \sum_{t=1}^{|N|} \sum_{u=1}^{|O|} -P(\omega_t, o_u) \log P(\omega_t, o_u) \tag{3.9}$$

The mutual information (MI) of variables N and O is defined as:

$$MI(N, O) = H(N) + H(O) - H(N, O) \tag{3.10}$$

Based on information theory, the mutual information measures the dependence of the two variables. Thus, if an encoding scheme is secure, the plaintext and the ciphertext should be independent of one another (*i.e.*, N and O are independent), which would yield MI$(N, O) = 0$. In general, lower MI indicates greater independence between the plaintext and the ciphertext and thus greater encoding security[‖].

3.3.5 Dataset

The NCVR dataset, described in Section 2.4.1, is used in this evaluation. We randomly select (without replacement) 100 datasets from NCVR, each containing 1,000 records, which

[‖]Note that *Edit Similarity* is an exception to this analysis because records are not encoded under this protocol.

we refer to as $A_1, \ldots, A_{100}$. The datasets $B_1, \ldots, B_{100}$ to which we link the aforementioned sets, are generated using the data corrupter, as described in Section 2.4.1.

The fields used are shown in Table 2.2. The fields *Street direction*, *Ethnicity*, and *Gender* are compared using a binary measure of similarity. Each of these fields consists of a single letter, so approximate field comparison does not add value.

When data is missing, the field receives a weight of 0, rather than the agreement or disagreement weight, and therefore does not factor into the record pair similarity score. This practice was also used by Gomatam *et al.* [48].

3.3.6 Implementation Details

Computer δ, as described in Table 2.1, is utilized for all experiments described in this chapter.

3.4 Results

In this section, we examine the results with respect to 1) correctness in record linkage, 2) computational complexity, and 3) security.

3.4.1 Correctness in Record Linkage

To ensure that the agreement and disagreement weights associated with each field are in line with intuition, Figure 3.6 depicts the weights from a randomly selected (A_z, B_z) linkage. As expected, the agreement weights are always higher than the disagreement weights, which means that the similarity score for a record pair increases when its fields match and decreases when its fields do not match. Also as expected, information-rich fields, such as *Surname*, are more important than information-poor fields, such as *Gender*, as indicated by the high agreement weight for *Surname*. While *Forename* is still important, it is not as discriminatory as *Surname*, as indicated by the lower agreement weight. This fits with intuition as there are 274,790 unique *Surnames* as compared to 187,743 unique

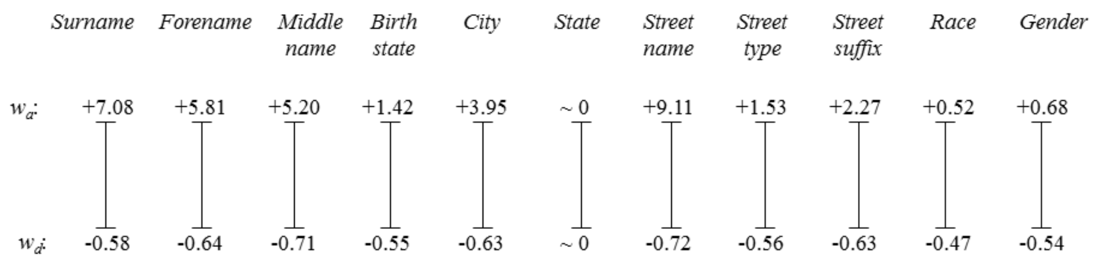


Figure 3.6: Agreement and disagreement weights for each field for one of the randomly sampled datasets.

Forenames in the NCVR database, indicating that *Surname* contains more information.

In Figure 3.7, the TP rate is reported for each string comparator. The *Bloom Filter* comparator performs best (TP rate = 0.9945) followed closely by the *Trigrams* (TP rate = 0.9883) and *Edit Similarity* (TP rate = 0.9854) comparators. The *Embedding* comparator does not perform as well as the others and achieves only a TP rate of 0.1384. It is interesting to note that all of the approximate PSCs, with the exception of the *Embedding* comparator, outperform the baseline PSC *Exact Agreement* (TP rate = 0.8397). Also of note, several of the approximate PSCs outperform the reference standard, *Jaro-Winkler* (TP rate = 0.9804). The implication of this result is discussed in Section 3.5.

3.4.2 Computational Complexity

It is also important to examine which comparators are computationally feasible for real record linkage applications. Figure 3.8 provides the running time of each field comparison method for comparing the fields in 10^6 record pairs. *Edit Similarity* required too much time to complete, so a smaller record linkage with files containing 10 records each (resulting in 100 pairs) was performed. The resulting running time, 74.09 ± 13.43 seconds, was then extrapolated to provide an estimate of the time required to compare the fields in 10^6 record pairs. In practice, this would take over two years to run and is clearly infeasible. Additionally, a time-consuming part of the protocol (specifically, an interactive cryptographic technique, known as 1-out-of-n oblivious transfer) was not included in this implementation

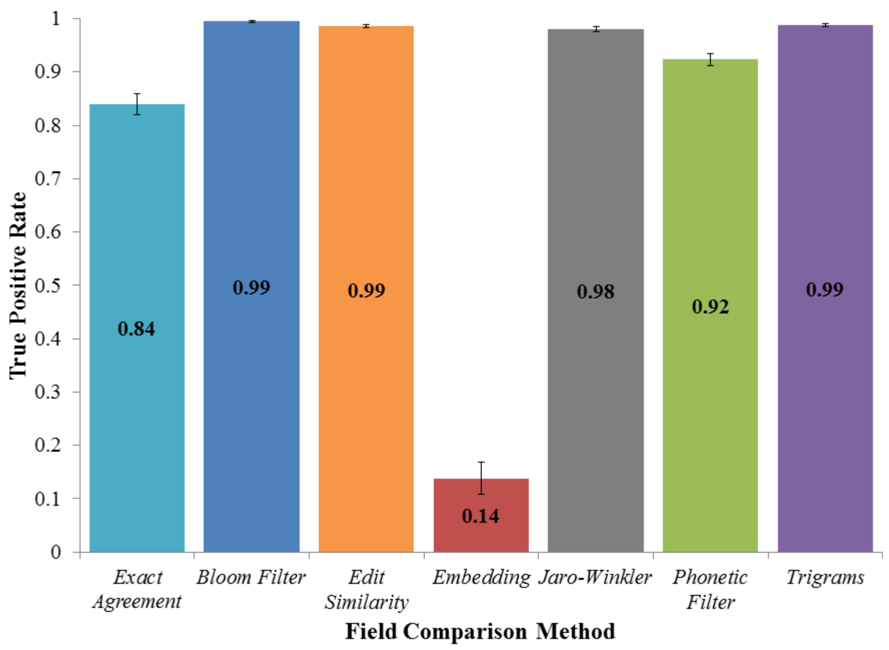


Figure 3.7: TP rate for private string comparators evaluated. The values reported are the averages over the 100 pairs of record files. Standard deviation error bars are shown.

and would only serve to further increase the running time.[**]

The running time of the *Embedding* comparator was an order of magnitude greater than the other comparators. The *Exact Matching*, *Jaro-Winkler*, and *Phonetic Filter* comparators all have comparable running times at around 330 seconds. The *Bloom Filter* is slightly slower at nearly 400 seconds followed by the *Trigrams* comparator at nearly 700 seconds.

3.4.3 Security

We examine the MI between the plaintext and the encoded form of the *Surname* values associated with each field comparator. These results are shown in Figure 3.9. The order of security as indicated by low MI is *Embedding* $>$ *Phonetic Filter* $>>$ *Bloom Filter* $\approxeq$ *Trigrams* which agrees with intuition (see qualitative discussion below).[††] Further details of the calculations are provided in Table 3.3. For reference, $H(\text{Plaintext}) = 8.9586$ and

[**]The values in Figure 3.7 were generated using an insecure, yet far quicker, version of edit similarity.
[††]The $>>$ symbol indicates "much greater than".

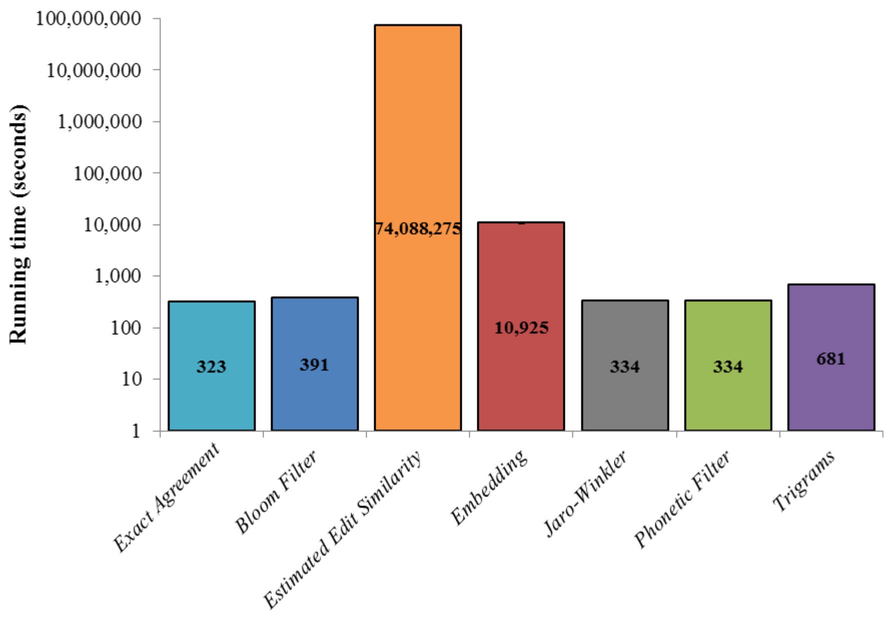


Figure 3.8: Field comparison running times for field comparators evaluated. The values reported are the averages, in log scale, over the 100 pairs of record files. Standard deviation error bars are shown.

there are 274,790 unique last names in the NCVR database.

String Comparison comparator Y	H(Y)	H(Plaintext,Y)	MI(Plaintext,Y)	# Unique Encodings
Bloom Filter	8.9581	0.0060	17.9106	274,430
Embedding	6.9008	0.0250	15.8344	55,097
Phonetic Filter	6.9214	0.0159	15.8641	5,815
Trigram	8.9586	0.0060	17.9111	274,790

Table 3.3: Entropy analysis.

As mentioned earlier, *Edit Similarity* is the most secure because the comparator does not encode the strings and does not require a third party Charlie. The only information revealed by this protocol is the length of strings in one's dataset, and this information is revealed only to the other data holder.

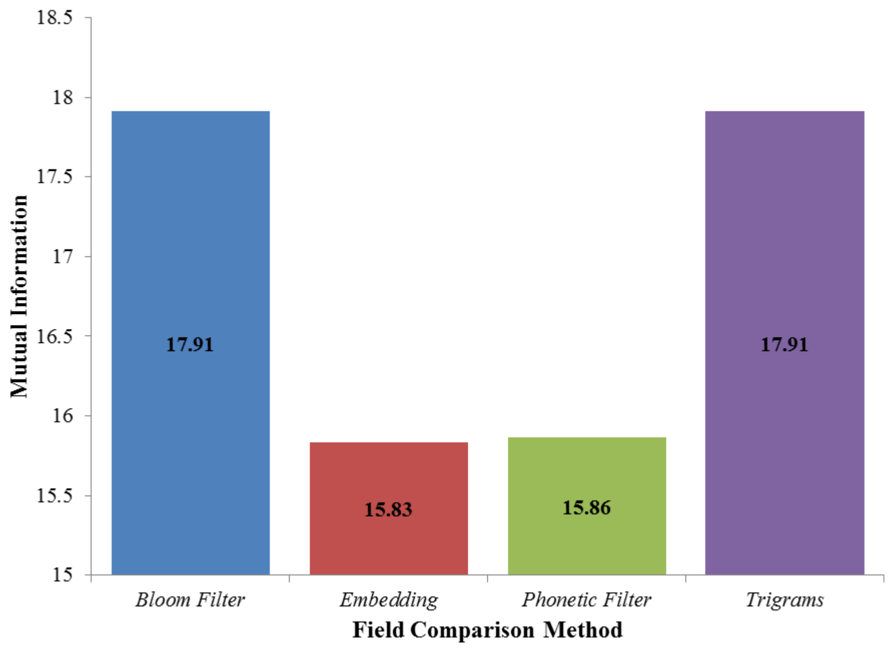


Figure 3.9: The mutual information between the plaintexts and ciphertexts associated with each field comparison comparator.

3.4.4 Summary of Results

The tradeoffs between correctness, computational complexity, and security are visualized in Figures 3.10, 3.11, and 3.12.

Figure 3.10 shows that the highly accurate comparators, (*Bloom Filter* and *Trigrams*), tend to be less secure. The *Embedding* comparator is more secure but not as accurate. An exception is the *Phonetic Filter* comparator which is both highly accurate and secure.

Figure 3.11 shows the general trend that the faster comparators are not as secure. Conversely, the *Embedding* comparator is very secure, but is also very slow. The *Phonetic Filter* comparator is an exception to this trend as it is both secure and fast when compared to the other comparators.

The tradeoff between correctness and computational complexity, shown in Figure 3.12, is more complex. The *Edit Similarity* comparator is highly accurate, yet is orders of magnitudes slower than the other metrics. Many of the other comparators achieve both

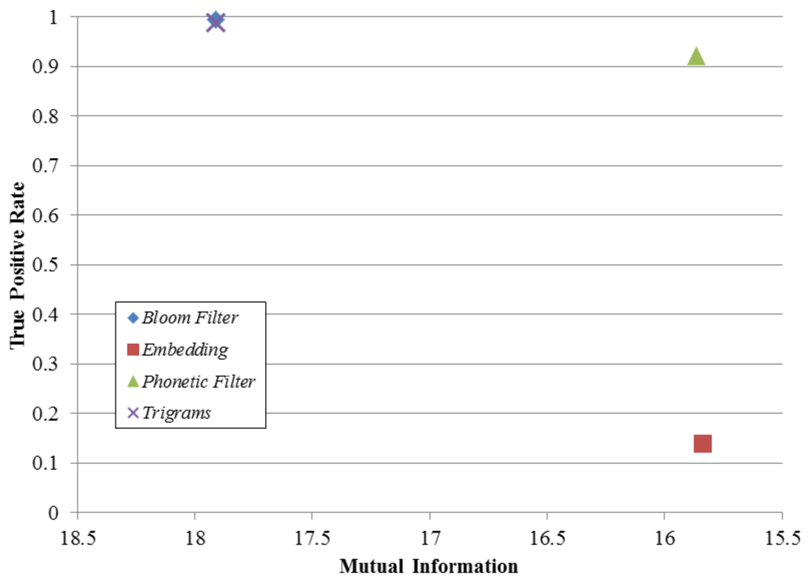


Figure 3.10: Tradeoffs between correctness, as measured by TP rate, and security, as measured by MI.

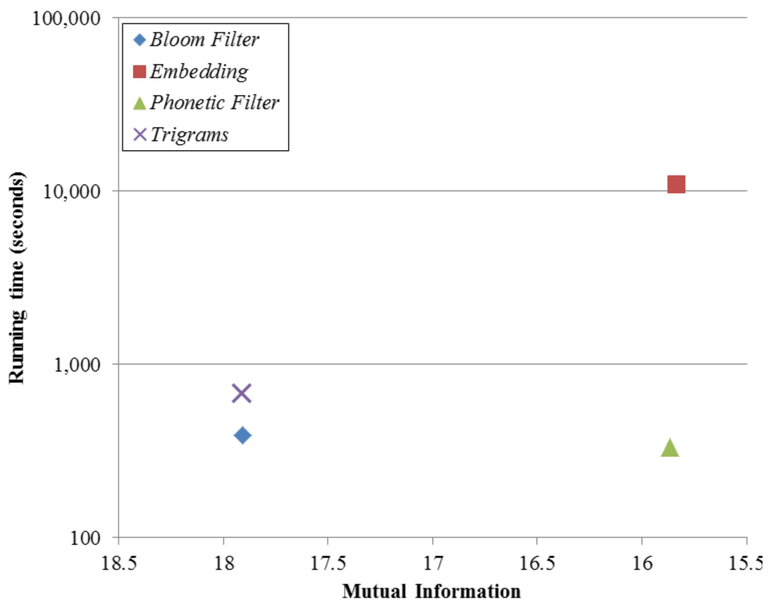


Figure 3.11: Tradeoffs between computational complexity (in log scale), as measured by running time, and security, as measured by MI.

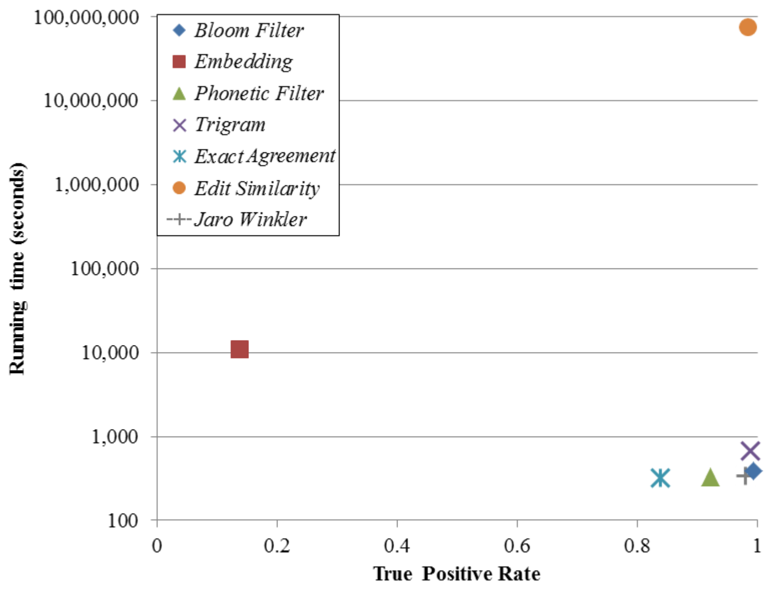


Figure 3.12: Tradeoffs between correctness, as measured by TP rate, and computational complexity (in log scale), as measured by running time.

high accuracy and expedient running times.

3.5 Discussion

In this section, we provide a summary and discussion of the results with respect to each of the three axes evaluated. Finally, we present additional considerations and limitations of this evaluation.

3.5.1 Correctness in Record Linkage

We measure correctness by TP rate, as defined in Equation 3.7. The *Exact Agreement* comparator achieves a TP rate of 0.8397. We believe it is able to correctly classify some record pairs due to the large number of fields (*i.e.*, eleven) included in each record. When there are many fields, even if several fields do not agree exactly, the number of additional fields that do increase the record pair's similarity score are enough to achieve a correct link prediction. We hypothesize that when fewer fields are included in each record, the ability to perform approximate string comparison will become even more important. This is because there will be fewer additional fields to increase the record pair's similarity score and push it over the classifying line. This idea is explored further in Chapter 4.

Perhaps surprisingly, several of the PSCs (*Bloom Filter*, *Trigrams*, and *Edit Similarity*) achieve higher TP rates than the plaintext reference standard, *Jaro-Winkler*. One explanation for this is that the data corrupter did not introduce errors in a way that *Jaro-Winkler* expects. The corrupter introduces errors uniformly throughout the string, whereas *Jaro-Winkler* expects to see fewer errors at the beginning. This may have handicapped *Jaro-Winkler*, and it may perform better on datasets that contain fewer errors at the beginning of strings. Additionally, *Jaro-Winkler* is particularly well-suited for *Names*, and in these experiments it was applied to non-name strings, such as *City* and *Street address*.

The *Embedding* approach did not perform as well as other PSCs. One reason for this may be the use of edit distance, rather than edit similarity, as the distance function. Another factor may be the use of randomly generated reference strings. If the reference strings are chosen more systematically such that they better cover the comparison space, the results might improve. While testing on a small sample indicated that using nine reference sets to define the embedding space is optimal (see Appendix A for further details), further calibration of this parameter may yield better results.

Although the *Bloom Filter* approach already performs very well, further improvements may be realized if the parameters (*i.e.*, filter length m and the number of hash functions k used) are tailored to the expected length of the strings in each field. For example, far fewer bits are set when the bigrams of the two-digit *State* abbreviation are hashed into a Bloom filter than when the bigrams of the longer field, *Street address*, are hashed. We test this hypothesis in Chapter 4.

The *Phonetic Filter* comparator is limited in that only the first several characters of a string are integrated into its encoding. In long or multi-token strings, such as "Lori Beth" (double name) or "Windy Hills" (street address), often the first token is the only one captured by this approach.

3.5.2 Computational Complexity

While *Edit Similarity* provides excellent security and high accuracy, its computational complexity renders it unrealistic for use in large scale PRL applications. The other comparators perform the relatively expensive encoding process once per record (2,000 times in the case of linking two 1,000-record datasets), and then perform an inexpensive record pair comparison process for each record pair (10^6 times). However, *Edit Similarity* does not include an up-front encoding process, so the expensive edit distance matrix calculation must be performed for *each* record pair. Since the number of record pairs is quadratic with respect to the number of input records, the running time of this approach quickly becomes intractable. Additionally, this comparator relies on a computationally-intensive form of encryption, a secure minimum finding protocol, which requires six expensive cryptographic operations for each cell of the matrix. Another expensive operation required by this protocol, 1-out-of-n oblivious transfer, was not implemented but would serve to further increase the running time.

The *Embedding* approach involves multiple steps. Generating the initial string embeddings and record pair comparison are relatively fast steps. However, the greedy resampling heuristic, during which the reference sets used are whittled down from all reference sets to a subset of "good" information-rich reference sets, is computationally expensive. If reference strings are selected with care so as to use information-rich reference strings (as opposed to randomly generated reference strings), the time-intensive greedy resampling step could be eliminated and the correctness of the results may improve.

A tradeoff between the running time and the storage space exists with the *Trigrams* comparator. While we used a space-efficient implementation, a vectorized implementation would yield reduced running times.

3.5.3 Security

Edit Similarity is the most secure in that the only information revealed to the other data holder is the string length. To prevent this exposure, all strings could be padded with an agreed-upon character up to the length of the longest string. This would prevent revelation of the string lengths and would not change the edit distance results, but it would increase running time. This would also prevent conversion of edit distance into the more informative measure of edit similarity.

As indicated by the MI values, *Embedding* and *Phonetic Filter* are the most secure and approximately equivalent. Little information is revealed by *Embedding* because Charlie is unaware of the strings defining the reference space, thus the embeddings provide little information. Each phonetically filtered string is a Soundex code consisting of a letter followed by three numbers, resulting in 26,000 possible Soundex codes. In practice, the 274,790 unique last names in the NCVR database map to only 5,815 phonetic filter encodings, resulting in information loss. Therefore, the *Phonetic Filter* approach also does not provide much information.

The *Bloom Filter* and *Trigrams* approaches are the least secure, as indicated by MI. In both of these comparators, salt is used to prevent a dictionary attack; however, these comparators are still subject to frequency analysis. The frequency distribution of n-grams is not as well-defined as the frequency distribution of *Forenames*, for example; however, an attacker may still be able to use this information to determine the values associated with some n-grams. The *Trigrams* comparator additionally reveals the length of each string to Charlie. He can derive this information based on the number of trigrams associated with each string (as the string length is the number of trigrams plus two, when the strings are padded with two spaces on both ends). The *Bloom Filter* comparator does not directly reveal the length of each string due to the fact that false positives and "crowding" (*i.e.*, the setting of a single bit multiple times) in the Bloom filter can occur, but it likely provides Charlie with the ability to bound the string length based on the number of bits set.

Additional work [76], performed jointly with collaborators, explores the extent to which this frequency information can be leveraged to determine plaintext values. A summary is that, for given parameter settings, a frequency-based cryptanalysis attack can reveal some plaintext values encoded in the Bloom filter. Work described in Chapter 4 describes how this vulnerability can be mitigated.

3.5.4 Additional Considerations

Firstly, these experiments were performed with relatively small datasets, but record linkage is often performed on a much larger scale. In such cases, blocking can used to reduce the number of record pair comparisons required [15]. We believe it is reasonable to consider each of the 1,000-record datasets analyzed in our experiments as representative of a single partition output from a blocking algorithm in a larger record linkage. To support this claim, we note that several private blocking methods exist (*e.g.*, [10, 62, 69]) but direct the reader to Chapter 5 for proper treatment of this topic.

Secondly, the goal of record linkage is to use the available information to determine which records refer to the same individual. Therefore in practice, fields used in record linkage may be numerical, rather than just string-based. Some of the PSCs considered are more extendable to numerical fields than others. We believe *Bloom Filter*, *Exact Matching*, and *Trigrams* should work well on numerical fields. If the distance function and embedding space were selected specifically for numerical fields, *Embedding* would also be applicable for numerical fields. The *Jaro-Winkler* and *Phonetic Filter* approaches are specifically tuned to work for strings and are therefore not applicable to numerical fields.

Thirdly, some of the comparators are more sensitive to the way in which information is partitioned into fields. For example, residential address can be represented as a single field, such as {*Street Address*}, or can be broken into more granular bits such as {*Street Number*}, {*Street Name*}, {*Street Suffix*}, and {*Street Direction*}. The *Exact Agreement* approach will perform best when fields are partitioned into segments because this approach cannot leverage the similarity information within a field. The *Phonetic Filter* and *Jaro-Winkler* approaches also perform better when fields are represented at more granular levels as both of these comparators rely on the tokens at the beginning of a string. The other approximate PSCs are less sensitive to the way in which information is divided into fields, but only to a certain extent. If too many fields are combined, one loses the ability to distinguish between more and less informative fields. For example, in the extreme case, all fields could be concatenated into a single field. In this case, less informative fields, such as *Gender*, cannot be distinguished from more important fields, such as *Surname*, and therefore one would expect classification accuracy to decrease.

Fourthly, one of the limitations of this research is that, though fields were selected to only contain strings, the inclusion of some numerical tokens was unavoidable. For example, although street number was excluded, sometimes the street name itself contains numbers (*i.e.*, "Highway 194"). Although this was very rare and efforts were made to include only string fields, this may have provided a slight advantage to string comparators more tolerant of numerical fields. In future research, we intend to determine how to best handle fields with a mixture of strings and numeric information.

3.6 Conclusions

In this chapter, we provide a principled and comprehensive evaluation of the state-of-the-art PSCs. The evaluation considered three axes critical to PRL applications: 1) correctness, 2) computational complexity, and 3) security. This research uses a real dataset, evaluates the PSCs on a common quantified scale, and provides information needed to support decisions in designing a PRL protocol in the real world. Using this analysis, a data manager can clearly model the tradeoffs and choose the best string comparator, based on his resource constraints. In summary, the *Bloom Filter* and *Trigrams* comparators are found to be the most accurate. The *Edit Similarity* comparator is most secure, followed by the *Embedding* and *Phonetic Filter* comparators. The *Edit Similarity* comparator proved to be computationally expensive, whereas the *Phonetic Filter*, *Exact Match*, and *Bloom Filter* comparators were the fastest PSCs.

3.6.1 Future Work

Based on the results of this evaluation, we select the *Bloom Filter* comparator for further evaluation. This work is described in Chapter 4.

CHAPTER IV

RECORD-LEVEL BLOOM FILTER ENCODINGS

In this chapter, we present a method for creating Bloom filter encodings that is more secure than existing approaches, yet maintains high accuracy in PRL.

4.1 Introduction

The evaluation presented in Chapter 3 identifies a promising encoding method, based on Bloom filters, for use in PRL. This encoding method yields highly accurate results while still running in a reasonable amount of time [42]. This method encodes field values in a Bloom filter in a way that enables an approximate measure of similarity [101]. However, recent work, performed jointly with collaborators, has questioned the security properties of the Bloom filter encodings [76] and has shown that, for certain parameter settings, a frequency-based cryptanalysis attack can reveal some plaintext values encoded in the Bloom filter.

In this chapter, we propose a method for encoding field values in a Bloom filter that is more secure and resistant to attack. Rather than encoding each individual field value in its own Bloom filter, we propose to encode all the field values in a single Bloom filter. We refer to field-level Bloom filter encodings as FBFs, and record-level Bloom filter encodings as RBFs. The principle behind the proposed method is that when each field is encoded individually, an attacker may utilize his knowledge about the frequency of values in this field to derive information about the encoded values. Our proposed data structure, a RBF, contains all field values for a given record, resulting in an encoding more resistant to frequency analysis. This is because the frequency distribution for the composition of fields is less distinct and less available, and therefore more difficult to use in a cryptanalysis attack (see discussion in Section 4.5.1 for further details).

The contributions of the work described in this chapter are the development of an encoding for use in PRL that has the following properties:

1. **Secure:** We present a more secure form of Bloom filter encodings for use in PRL such that all field values for a record are encoded in a single Bloom filter. These RBF encodings are resistant to an attack known to compromise FBFs, as proposed in the literature [76].

2. **Distance-preserving:** The RBFs presented in this chapter provide for a distance-preserving transformation from the plaintext space to the ciphertext space such that records similar to one another in the plaintext space are also similar to one another in the ciphertext space. This paves the way for the application of PRL in the ciphertext space to identify records that correspond to the same individual.

4.1.1 Chapter Outline

The remainder of this chapter is organized as follows. In Section 4.2, we summarize the FBFs proposed in the literature as well as the results of our recent evaluation and security analysis of this encoding method. Section 4.3 describes several methods we have developed for creating a RBF encoding. This section also describes the experiments we use to systematically appraise the accuracy and security of RBFs, the evaluation metrics used to measure these properties, and the controls we use in these experiments. In Section 4.4, we present the results of these experiments to measure the accuracy and security of the RBFs. In Section 4.5, we discuss the practical implications of these results, and in Section 4.6, we conclude and summarize this chapter.

4.2 Related Work

A common method for performing PRL is to encrypt each field value [7, 8, 18, 45, 94] and send the encodings to a centralized third party, who then performs record linkage by comparing the encoded field values [34, 94]. A drawback to the use of traditional encryption methods is the loss of the ability to detect field similarity in the encrypted space. This stems from the fact that secure encryption methods are not similarity-preserving transformations. Therefore, the transformation from plaintext to ciphertext can cause a loss of information, and records that exhibit high similarity in the plaintext space may be quite dissimilar in the ciphertext space. As a result, the encoded records may be incorrectly classified as *Predicted Non-matches*.

To address this drawback, the encoding methods evaluated in Chapter 3 [12, 32, 40, 60, 63, 68, 70, 71, 92, 95, 99, 101, 103] enable similarity detection in the ciphertext space. The ability to measure the approximate similarity of encoded values means that the transformation from plaintext to ciphertext is not as lossy. Therefore, encodings which provide for approximate field comparison have the potential to improve record linkage accuracy. In fact, the evaluation presented in Chapter 3 and published in [41, 42] shows this to be the case.

4.2.1 Field-level Bloom Filter Encodings

The results of the evaluation in Chapter 3 also highlight a method, based on Bloom filter encodings, that provides highly accurate record linkage results while still running in a reasonable amount of time [42]. To review, a summary of how to generate encodings according to this technique [101] is shown in Figure 3.2 and consists of the following steps.

- Each string is padded with $n - 1$ blank characters on each end.
- The set of n-grams for each string is then computed.
- Each n-gram is hashed into a Bloom filter of length m bits using k hash functions.

The encoded values can be compared using a set-based similarity measure, such as the Dice coefficient (Equation 3.2), in a way that yields an approximate measure of similarity [101]. This encoding method provides a foundation for the more secure encoding that we propose in this chapter.

4.2.2 Security Flaw of FBFs

Recall that in Section 3.4.3, the security of several PSCs is measured by the amount of mutual information (described in Section 3.3.4) shared between the plaintext and ciphertext values. The idea is that the plaintext and ciphertext values should be indendent of one another, and therefore have a low mutual information value. These results showed that FBFs were not as secure as some other encoding methods.

This result led to additional work to further examine the security properties of FBFs [76], where the goal was to determine how much information about plaintext values could be revealed by a cryptanalysis of the encoded ciphertext values. Specifically, the question of interest is "How much information can Charlie gain about the sensitive plaintext data by examing the encoded data?" Recall that Charlie is the third party who receives the encoded values from Alice and Bob and performs PRL using these encoded values.

The cryptanalysis made the following assumptions about the information Charlie has.

Assumption 4.1 *Charlie knows the global dataset from which Alice's and Bob's records are drawn.*

Assumption 4.2 *Charlie knows the number of hash functions (k) that are part of the encoding function.*

We believe Assumption 4.1 is reasonable because record linkage is typically performed with demographic information, which is often publicly available. With respect to Assumption 4.2, Charlie can infer k by examining the number of bits set in the Bloom filters. More specifically, due to hash collisions, Charlie can only estimate this number, but it can be restricted to a small range, such that the proposed attack can be applied for all values in the range.

A common method for cryptanalysis attacks is based upon examination of the frequencies of values seen in the ciphertext. Once these frequencies are tallied, one can attempt to map them back to the frequency values seen in the plaintext. In this specific attack, we observe the frequencies with which bits are set in the Bloom filter encodings and attempt to map bits back to the plaintext n-grams to which they correspond. This mapping of frequencies is modeled as a constraint satisfaction problem (CSP).

The results of this analysis show that, for some parameter settings, Bloom filter encodings are vulnerable to attack, and that Charlie can successfully execute a frequency-based cryptanalysis to reveal some plaintext values. However, as n grows, the attack becomes less successful. Therefore, the analysis also demonstrated that Bloom filter encodings based on trigrams or quadgrams, rather than bigrams, are more resistant to this type of attack.

However, as n grows, accuracy declines. In this work, we explore more secure encoding methods to further increase security and to apply in cases where a large n value is not suitable.

4.3 Materials & Methods

The results of the PSC evaluation presented in Chapter 3 (which were recently published [42]) and the cryptanalysis of Bloom filter encodings [76] led us to develop a novel encoding method. The proposed method builds upon the strengths of the FBFs, such as its high accuracy and expedient running time, while improving upon its security properties. In Section 4.3.1, we present two changes we make to FBF encodings to generate a more secure encoding. Section 4.3.2 describes the control measures that serve as a reference standard. Section 4.3.3 presents metrics used to evaluate the encodings, and Section 4.3.4 describes the dataset used in this chapter.

4.3.1 Experimental Methods

We introduce two improvements to increase the security of FBF encodings. The first is based upon consolidating FBFs into a single RBF, and the second is based upon selecting parameters for FBF creation to improve the security properties.

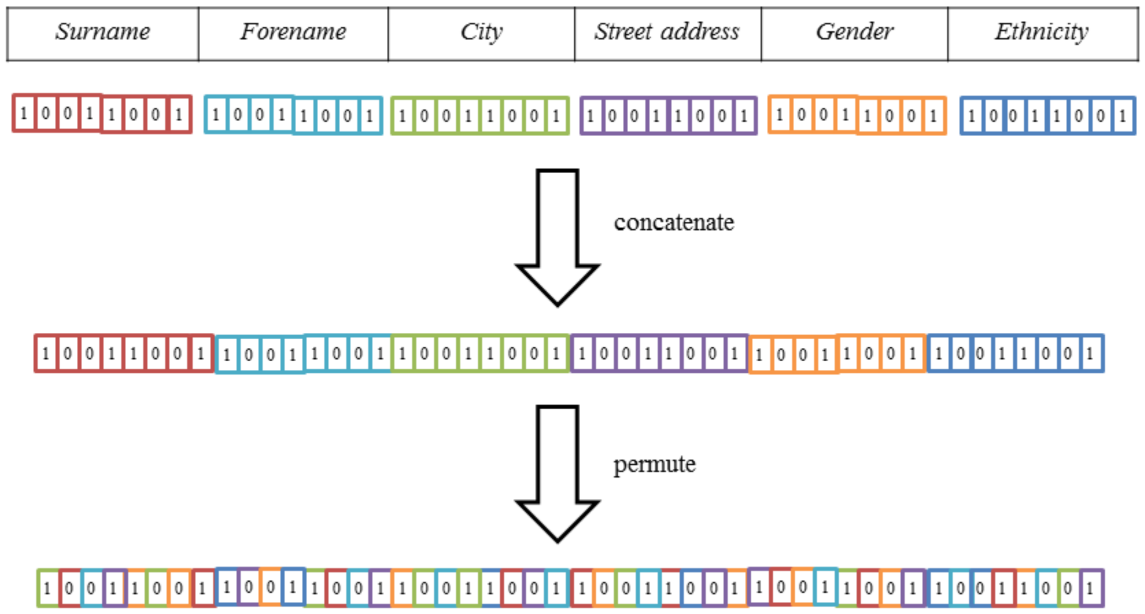


Figure 4.1: Record-level Bloom filter generation. Bits are sampled from each FBF to create a RBF. The bits are then permuted to increase security.

RBF generation

Rather than encoding each field value in its own Bloom filter as proposed in [101], we recommend that a single Bloom filter be used for encoding the entire record. We hypothesize that this composite encoding, when carefully designed, provides greater security. We now describe, at a high level, how to create a RBF, given the FBFs corresponding to the fields in the record.

The big picture behind this approach is that bits are sampled from each FBF and concatenated to create a single RBF for the record. The bits in the RBF can then be permuted such that it is unclear from which field a given bit was drawn. The general architecture for this approach is illustrated in Figure 4.1.

Two questions must be addressed to guide RBF generation:

1. *How many* bits should be drawn from each FBF for inclusion in the RBF?

2. *Which* bits should be sampled from the FBFs for inclusion in the RBF?

Both of these questions are answered in the remainder of Section 4.3.1.

Guidance on *how many* bits should be sampled from each FBF

We now describe two methods for determining how many bits should be sampled from each FBF. The first is based upon *unifom* bit selection, and the second proposes a more intelligent manner for *weighted* bit selection based on the discriminatory power of the field. We hypothesize that *weighted* bit selection will provide RBF encodings that yield more accurate record linkage results.

Uniform bit selection

In the most straightforward case, bits are sampled uniformly from each FBF to create the RBF such that an equivalent number of bits are drawn from each field. For example, if the goal is to create a RBF of length 1,000 bits, and each record has 10 fields, then 100 bits are sampled from each FBF.

While this approach is very simple, when bits in the RBF are permuted, it removes some of the information that an attacker can leverage to determine the corresponding encoded field values. Specifcally, if an attacker does not know which bits correspond to which field, it becomes more difficult to leverage the knowledge of the frequency at which certain field values appear.

Weighted bit selection

We hypothesize that sampling bits based on the discriminatory power of each field will provide a RBF that will yield more accurate record linkage results than the *uniform* bit sampling strategy described above.

We now discuss methods for determining the weight associated with each field. Data holders Alice and Bob may already have an idea of the discriminatory power of each field, and can weight each field accordingly. However, in the absence of this knowledge, there are methods to determine the weightings for each field. One such method is the FS field weighting approach [46] described in Section 2.1.6. This method outputs an agreement weight and a disagreement weight associated with each field. In traditional record linkage

	Surname	*Forename*	*City*	*Street*	*Ethnicity*	*Gender*
m	0.54	0.58	0.57	0.62	0.63	0.66
u	5.5x10^{-4}	1.5x10^{-3}	7.9x10^{-3}	1.7x10^{-4}	3.6x10^{-1}	3.1x10^{-1}
w_a	6.9	6.0	4.3	8.2	0.6	0.8
w_d	-0.8	-0.9	-0.8	-1.0	-0.5	-0.7
range	7.7	6.9	5.1	9.2	1.1	1.5
w	24%	22%	16%	29%	4%	5%

Table 4.1: Weighted RBF creation. FS weights can assist in determining the extent to which each FBF should contribute to the RBF.

applications, these weights are used in the record comparison step (see Section 2.1.7) to calculate a similarity "score" for each record pair.

We propose a new application for these weights. In Equations 4.1 and 4.2, we present a means for combining these agreement and disagreement weights into a single weight, w, associated with each field i.

$$range[i] = w_a[i] - w_d[i] \tag{4.1}$$

$$w[i] = \frac{range[i]}{\sum_{i=1}^{f} range[i]} \tag{4.2}$$

This weight is a measure of the relative discriminatory power of each field, as compared to other fields. We propose this weight be used to determine how many bits are drawn from each FBF for inclusion in the RBF. The weight associated with each field is derived by dividing by the sum of the ranges over all fields (*i.e.*, normalizing) to determine the percentage of bits in the RBF that should be drawn from that field. The m, u, w_a, w_d, *range*, and w values for the dataset used in this chapter (see Section 4.3.4 for additional details) are shown in Table 4.1.

Returning to our example in Figure Table 4.1, it can be seen that by applying this approach, we sample a varying number of bits from each field. For instance, the *Surname* and *Forename* fields comprise 24% and 22% of the bits to the FBF, respectively.

Setting the m_{RBF} value

	Surname	*Forename*	*City*	*Street*	*Ethnicity*	*Gender*
# bits in FBF	100	100	100	100	100	100
% RBF composition	24%	22%	16%	29%	4%	5%
Possible m_{RBF} values	415	455	625	345	2,500	2,000

Table 4.2: m_{RBF} determination.

	Surname	*Forename*	*City*	*Street*	*Ethnicity*	*Gender*	Sum
RBF % composition	24%	22%	16%	29%	4%	5%	100%
RBF composition	600	550	400	725	100	125	2,500

Table 4.3: An example of RBF composition.

Given a strategy for determining RBF composition, in order to determine the actual number of bits to be drawn from each FBF, we must select a value for the number of bits to be included in the RBF, which we call m_{RBF}. To provide a baseline, we assume that all information (*i.e.*, bits) from each field should be included in the RBF. Given this assumption and the weight associated with each field, we can calculate the necessary value for m_{RBF} if each field is included in full and at the relative discriminatory power calculated. We take the maximum of these values to be m_{RBF}. An example is shown in Table 4.2 where the length of each FBF is 100 bits. If the 100-bit long FBF for the field *Ethnicity* is to comprise 4% of the record-level Bloom filter, then m_{RBF} must be 2,500 bits. As this is the maximal value for m_{RBF} required across all fields, this is selected as m_{RBF}. The number of bits to be drawn from each FBF is then selected in accordance with field weight w, and the final composition of the RBF is shown in Table 4.3.

To adhere to the selected percent compositions, some bits must be resampled for inclusion in the RBF. For example, 600 bits from the FBF encoding of *City* are to be included in the RBF; however, the FBF encoding of *City* only has 100 bits. This implies that these bits will be included multiple times in the RBF.

This method selects the value m_{RBF} based on the assumption that all bits from each

FBF should be included in the RBF. To check whether this assumption is reasonable, we perform a "condensation analysis" by progressively condensing the RBF. In the experiments below, we consider m_{RBF} values in $\{50; 100; 500; 1{,}000\}$.

Guidance on *which* bits should be sampled from each FBF

While we have discussed methods to determine *how many* bits should be sampled from each FBF, *which* bits should be sampled remains unaddressed. We propose random bit sampling, with repetition, for the following reasons:

1. Any strategy would give an adversary knowledge of which bits are included in the RBF and thus better knowledge of how to attack the data structure.

2. Recall that k hash functions are used to hash each n-gram into the FBF. This results in a non-uniform bit distribution in which some bit values are highly correlated. This conditional dependency must be taken into account when determining which bits are informative and should be included in the FBF. This would make any intelligent sampling strategy complex.

3. As PRL can be a computationally intense process, an efficient encoding function is important. For example, some secure multi-party computation techniques, such as the *Edit Distance* approach evaluated in Chapter 3, provide excellent security guarantees, but are too computationally intense for real world use. A benefit of the Bloom filter-based encodings is the speed with which they can be generated and compared. Random bit sampling, in contrast to a more intelligent strategy, allows us to maintain the efficiency of the encoding.

Throughout the remainder of this work, random bit sampling, with replacement, is used in RBF generation.

FBF sizing

We now discuss how FBF parameters can affect encoding security and draw a distinction between *static* and *dynamic* FBF sizing. Recall from Section 3.2.2 that the parameters involved in Bloom filter encoding are n, the size of the substring used in string tokenization, k, the number of hash functions used for each n-gram, and m, the number of bits in the Bloom filter. We let m_{FBF} refer to the number of bits in each FBF.

Static FBF sizing

When FBFs are statically sized, the k and m values are held constant across all fields, regardless of the expected number of n-grams to be encoded in the field. In this case, observation of the percentage of bits set may allow an attacker to determine some information about the length of the encoded value and the number of hash functions, k, used in encoding.

Shorter fields, such as *Gender*, result in Bloom filter encodings with fewer bits set than longer fields, such as *Forename.* Consider an example where n=2, k=15, and m=500. *Gender*, encoded "M" or "F", will contain only two n-grams for field value (*i.e.*, " M" and "M " for the male gender). Therefore, for each gender encoding, a maximum of 30 bits can be set (15 per n-gram), which corresponds to 6% of the bits in the Bloom filter. However, if *Forename* has an average of 6 letters, this corresponds to 7 n-grams when $n = 2$, and a maximum of 105 bits can be set, which corresponds to 21% of the bits in the Bloom filter.

As a consequence, when static parameters are used across fields, an attacker can observe the percentage of bits set in a FBF. With this information, an attacker can determine information about the length of the value encoded in the Bloom filter and the number of hash functions, k, used in encoding.

Dynamic FBF sizing

Recall that, in Section 3.5.1, the suggestion was made to tailor encoding parameters to the specific properties of each field, rather than use static encoding parameters across all fields. In order to limit the extent to which an attacker can determine information about the field from which a bit was drawn, we propose *dynamic*ally sizing the Bloom filters (*i.e.*, setting the m value) such that we expect the same percentage of bits to be set in each FBF across all fields. Additionally, we let the expected frequency at which a given bit is set in

	Surname	*Forename*	*City*	*Street*	*Ethnicity*	*Gender*
Average field length	6.3	5.8	8.8	8.7	1.0	1.0
Average # n-grams (g)	7.4	6.8	9.8	9.7	2.0	2.0
m_{RBF}	159	149	214	212	44	44

Table 4.4: Example of *dynamic* Bloom filter sizing where k=15, n=2, p=0.5.

the Bloom filter encoding equal 0.5. In order to maximize entropy, and therefore maximize security, half of the bits should be set and half of the bits should remain unset [97]. In this case, an attacker would see approximately half of bits set and would gain little information about the field values encoded in the Bloom filter.

More formally, let p be the expected frequency at which a bit is set and let g be the number of items (*e.g.*, n-grams) stored in the Bloom filter. Therefore, let $p = 0.5$.

The work by Roozenburg [97] states that if g elements have been stored in a Bloom filter, the probability that a certain bit remains unset (*i.e.*, a value of 0) is $p = 1 - \frac{1}{m}^{kg}$. Therefore, we select a value for k, and hold it constant, we can calculate the number of bits needed in the Bloom filter according to Equation 4.3. An example of the application of this principle is shown in Figure 4.4.

$$m = \frac{1}{1 - \sqrt[kg]{p}} \tag{4.3}$$

Using this approach, we can measure the information content associated with each field and adjust the Bloom filter length accordingly.

We hypothesize that RBFs composed of *dynamic*ally-sized FBFs will be more secure than RBFs composed of *static*ally-sized FBFs. This is because *dynamic* sizing means an observer will not be able to determine information about the length of the encoded values by observing the percentage of bits set in the FBF.

RBF Variations

This chapter has introduced two ways to determine the composition of bits in the RBF using either *uniform* bit selection or *weighted* bit selection. Additionally, this chapter has

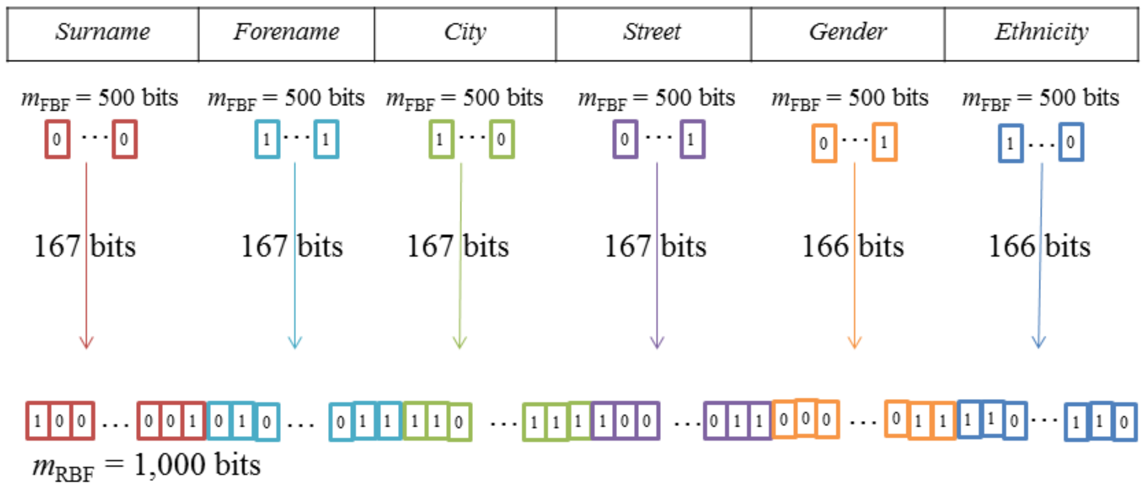


Figure 4.2: *Static/Uniform* RBFs are generated by sampling uniformly from *static*ally sized FBFs.

proposed two methods for determining the FBF encoding parameters – *static* parameters regardless of expected field length, and *dynamic* parameters based on the expected length of the field to be encoded. In this section, we describe all four variations of RBFs that result from combining the two possibilities for FBF sizing and bit selection. The naming convention for the different variations is *method for FBF sizing / method for bit selection.*

For illustration, throughout this section, we assume there are six fields and the length of the RBF (referred to as m_{RBF}) is 1,000 bits. We discuss methods for determining m_{RBF} in Section 4.3.1. Also, throughout this section, we let $k = 15$ as recommended by Schnell *et al.* [101].

<u>*Static/Uniform* RBFs</u>

For the *Static/Uniform* RBF, the FBFs are sized statically, such that $m = 500$ as recommended by Schnell *et al.* [101]. In this example, there are six fields, $1,000/6 \approx 167$ bits are drawn from each FBF*. An example is shown in Figure 4.2.

<u>*Static/Weighted* RBFs</u>

For the *Static/Weighted* RBF, the FBFs are sized statically such that $m_{\text{FBF}} = 500$, and again m_{RBF} is 1,000 bits. The number of bits to be sampled from each field is determined

*In some cases, rounding is required if m_{RBF} is not evenly divisible by the number of fields.

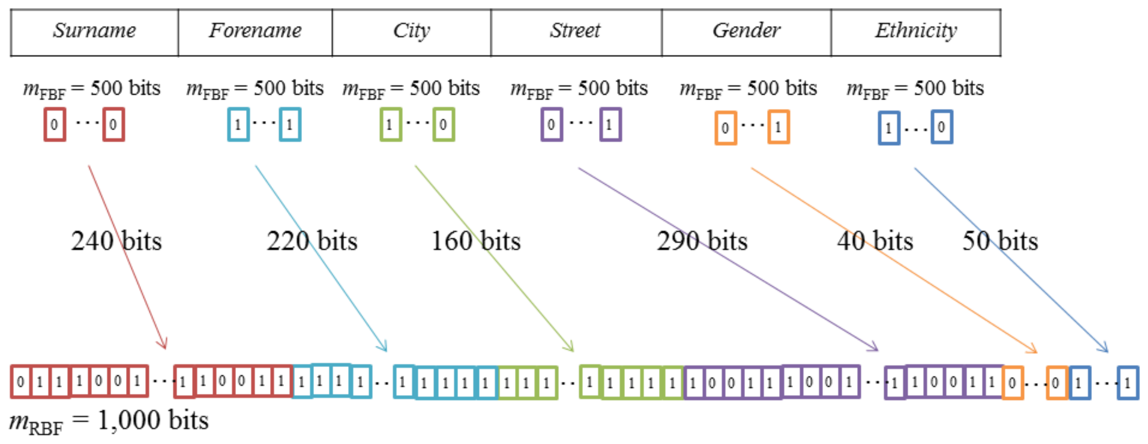


Figure 4.3: *Static/Weighted* RBFs are generated by sampling, according to a weight representing the discriminatory power of each field, from *static*ally sized FBFs.

by multiplying the percentages shown in Table 4.1 by the total length of 1,000 bits. An example is shown in Figure 4.3.

Dynamic/Uniform RBFs

In this instance, FBFs are sized dynamically according to the parameters shown in 4.4. Bits are sampled uniformly from each field, so ∼167 bits are taken from each FBF. This is illustrated in Figure 4.4.

Dynamic/Weighted RBFs

In this instance, FBFs are sized dynamically according to the parameters shown in 4.4. The number of bits to be sampled from each field is determined by multiplying the percentages shown in Table 4.1 by the total length of 1,000 bits. An example is shown in Figure 4.5.

As previously mentioned, to provide a baseline, we assume that all information from each field should be included in the RBF. Tables 4.5 and 4.6 show the details of m_{RBF} determination for *Dynamic/Weighted* RBFs. If the 214-bit long FBF for the field *City* is to comprise 16% of the record-level Bloom filter, then m_{RBF} must be 1,315 bits. As this is the maximal value for m_{RBF} required across all fields, this is selected as the length for the RBF. The number of bits to be drawn from each FBF is then selected in accordance with the field weight, as calculated in Table 4.1 and Equation 4.1. The final composition of the RBF is shown in Table 4.6.

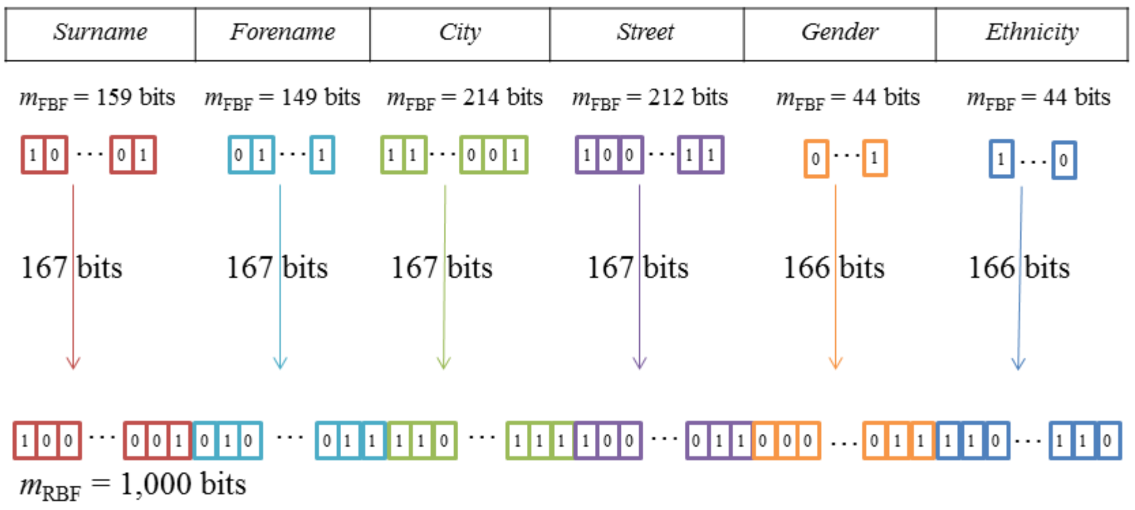

Surname
Forename
City
Street
Gender
Ethnicity
m_{FBF} = 159 bits
m_{FBF} = 149 bits
m_{FBF} = 214 bits
m_{FBF} = 212 bits
m_{FBF} = 44 bits
m_{FBF} = 44 bits
167 bits
167 bits
167 bits
167 bits
166 bits
166 bits
m_{RBF} = 1,000 bits

Figure 4.4: *Dynamic/Uniform* RBFs are generated by sampling uniformly from *dynamic*ally sized FBFs.

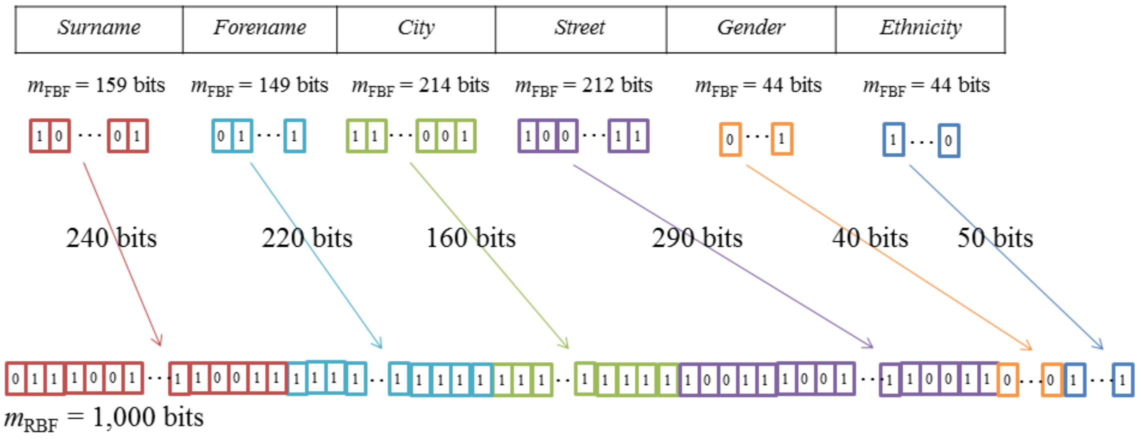

Surname
Forename
City
Street
Gender
Ethnicity
m_{FBF} = 159 bits
m_{FBF} = 149 bits
m_{FBF} = 214 bits
m_{FBF} = 212 bits
m_{FBF} = 44 bits
m_{FBF} = 44 bits
240 bits
220 bits
160 bits
290 bits
40 bits
50 bits
m_{RBF} = 1,000 bits

Figure 4.5: *Dynamic/Weighted* RBFs are generated by sampling, according to a weight representing the discriminatory power of each field, from *dynamic*ally sized FBFs.

	Surname	*Forename*	*City*	*Street*	*Ethnicity*	*Gender*
m_{FBF}	159	149	214	212	44	44
% RBF composition	24%	22%	16%	29%	4%	5%
Possible m_{RBF} values	651	683	1,315	726	1,245	934

Table 4.5: m_{RBF} determination for *Dynamic/Weighted* RBFs.

	Surname	*Forename*	*City*	*Street*	*Ethnicity*	*Gender*	Sum
RBF % composition	24%	22%	16%	29%	4%	5%	100%
RBF composition	321	287	214	384	46	62	1,315

Table 4.6: An example of RBF composition for *Dynamic/Weighted* RBFs.

4.3.2 Controls

The following field comparison techniques are used as controls for the experimental RBF techniques:

1. Binary field comparison, in which field values are checked for equality, resulting in a binary measure of similarity, and
2. The plaintext JW string comparator, as defined in Section 3.3.3.

The controls are used to determine field similarity. To convert these field similarities to record pair similarities, the field similarities are multiplied, as indicated below, by either a uniform value ($\frac{1}{f}$) or the weighted percentages calculated in Table 4.1.

4.3.3 Evaluation Metrics

"Neighborhood test" to measure accuracy

To determine the extent to which proximity is preserved in the transformation to the encoded space, we developed the "Neighborhood test". In this experiment, we sample a subset of the records and measure the similarity of each sample to each record in the dataset to which the sample is to be linked. We then rank order all records in the other dataset by their similarity to the sample record. The index in this rank ordered list at which the sample record's *True Match* falls is called the neighbor rank for the sample record. If the transformation preserves similarity well, the *True Matches* should be very close to one another and therefore close neighbors. In the best case, a sample's neighbor rank is 1 (*i.e.*, the sample is closest to its *True Match*), and in the worst case, the sample's neighbor rank is the number of records in the opposing dataset (*i.e.*, the sample's *True Match* is the most distant record in the dataset to which the sample is being linked).

Frequency analysis to examine security

An attacker may try to map bits in the RBF back to the fields from which they were drawn. If an attacker is successful, he could then mount an attack such as the one described in Section 4.2.2, using the FBFs as inputs. Therefore, evaluating the extent to which an attacker can map bits in the RBF back to the field from which they were drawn is important to characterize the security of the RBF.

To perform a frequency analysis, the encodings are examined and the frequencies at which bits are set are calculated over these encodings. In this case, the frequencies are calculated over all of the encoded NCVR records, both clean and corrupt, as described in Section 4.3.4.

4.3.4 Dataset

For these experiments, we use 100,000 records from the NCVR dataset. A second set of records to which the original is linked is generated using a data corrupter as described in Section 2.4.1. We exclude records with missing field values to obtain a clear idea of the performance of each method without this complicating factor. We address handling missing values in the context of Bloom filter encodings in Section 6.3.1.

4.3.5 Implementation Details

Computer ϵ, as described in Table 2.1, is utilized for all experiments described in this chapter.

4.4 Results

4.4.1 Frequency Analysis Results

In this section, we examine how much information an attacker could obtain by analyzing the frequencies with which bits in the RBFs are set. The average, taken over all bits within a field, frequency at which bits within each field are set is shown in Figure 4.6.

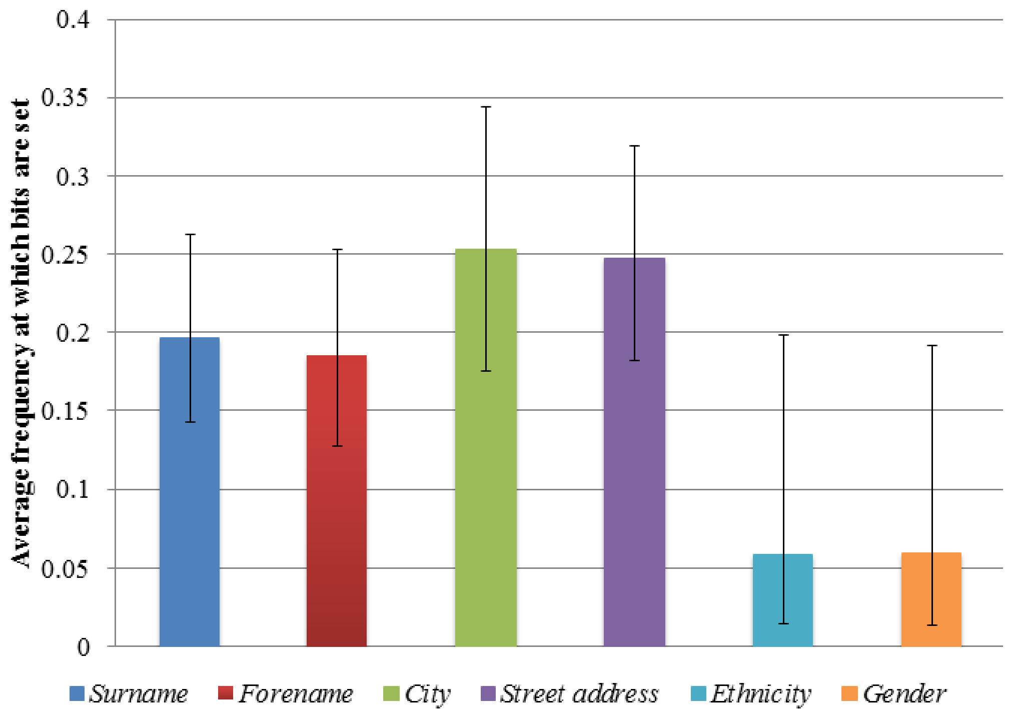
The FBFs of *dynamic* size have a more uniform average frequency across fields than the FBFs of *static* size. This suggests that it may be easier for an attacker to determine from which FBF a bit was drawn if the FBFs are statically sized.

Next, we look beyond the averages to more detailed information about the frequency with with each bit is set. Figure 4.7 shows a heat map of the frequencies at which bits are set across all NCVR encodings in FBFs of *static* size. Figure 4.8 shows these values for the FBFs of *dynamic* size. Figures 4.7 and 4.8 indicate that the frequency distributions for categorically-valued fields (Figures 4.7(e), 4.7(f), 4.8(e), 4.8(f)) are less uniformly distributed than for string-based fields (Figures 4.7(a) - 4.7(d), 4.8(a) - 4.8(d)). Due to the fact that categorical variables can take on a limited number of values, many of the bits in the Bloom filter are never set. Those bits that are set, are generally set frequently, because many people in the population may have that value.

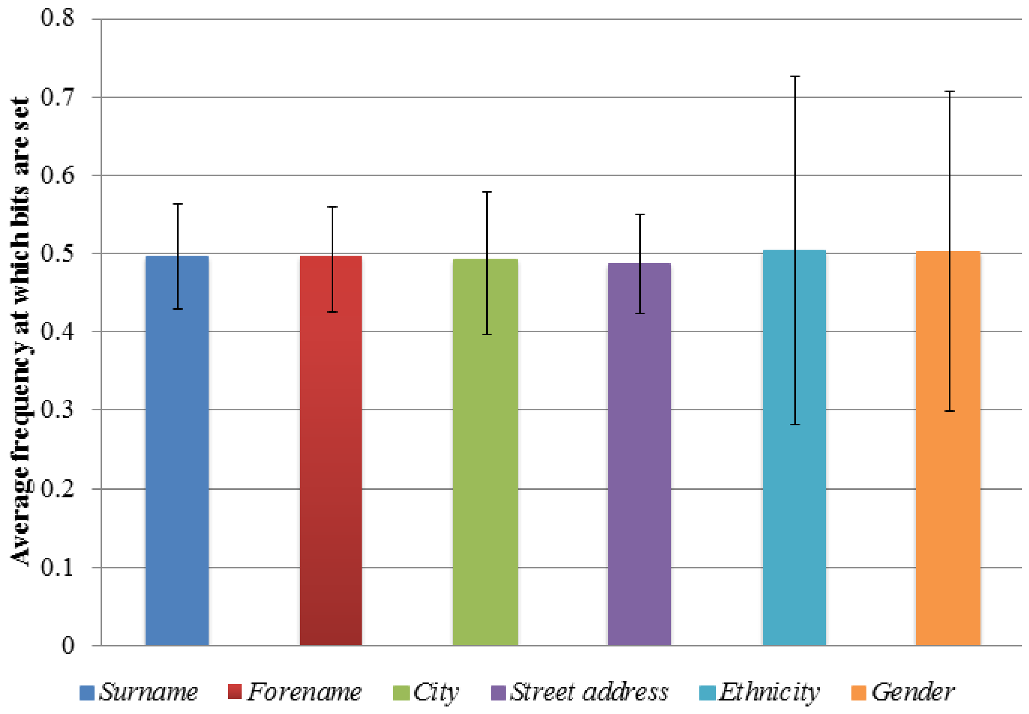
In general, the evidence illustrates that FBFs of *dynamic* size are more uniformly distributed than the FBFs of *static* size. This finding indicates that an attacker would have greater difficulty using this frequency information in an attack. Therefore, we focus on RBFs composed of *dynamic*ally sized FBFs in the remainder of this chapter.

4.4.2 Neighborhood Test Results

We use the Neighborhood test to measure the accuracy afforded by different encoding methods. An ideal encoding technique preserves distance in the transformation from plaintext to ciphertext such that the record nearest a sample record is its *True Match*, resulting in a neighbor rank value of 1. Figure 4.9 presents the results of the Neighborhood test. Notably,

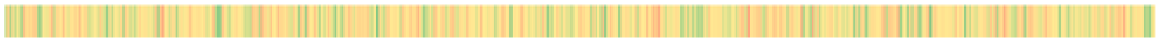


(a) *Static* FBFs.

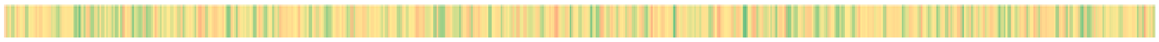


(b) *Dynamic* FBFs.

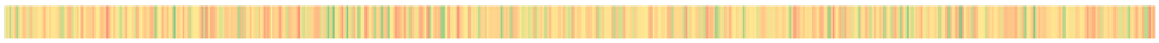
Figure 4.6: The effect of FBF sizing on the average percentage of bits set. Standard deviation error bars are shown. The probabilities for *Static* FBFs are shown in part (a) and for *Dynamic* FBFs in part (b).



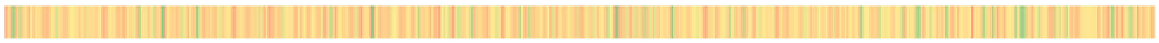
(a) Surname.



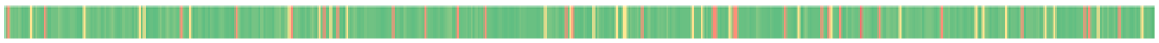
(b) Forename.



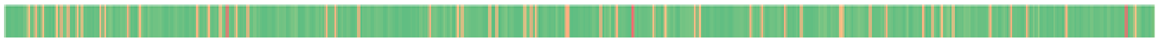
(c) City.



(d) Street address.

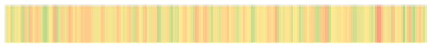


(e) Ethnicity.

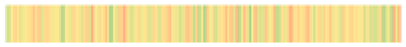


(f) Gender.

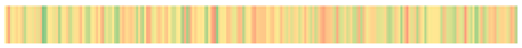
Figure 4.7: A heat map representation of the frequencies at which bits in the FBFs of *static* size are set. In all cases, $m_{FBF} = 500$.



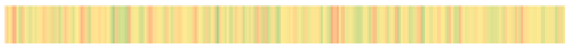
(a) Surname, $m_{FBF} = 159$ bits.



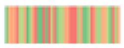
(b) Forename, $m_{FBF} = 149$ bits.



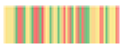
(c) City, $m_{FBF} = 214$ bits.



(d) Street address, $m_{FBF} = 212$ bits.



(e) Ethnicity, $m_{FBF} = 44$ bits.



(f) Gender, $m_{FBF} = 44$ bits.

Figure 4.8: A heat map representation of the frequencies at which bits in the FBFs of *dynamic* size are set.

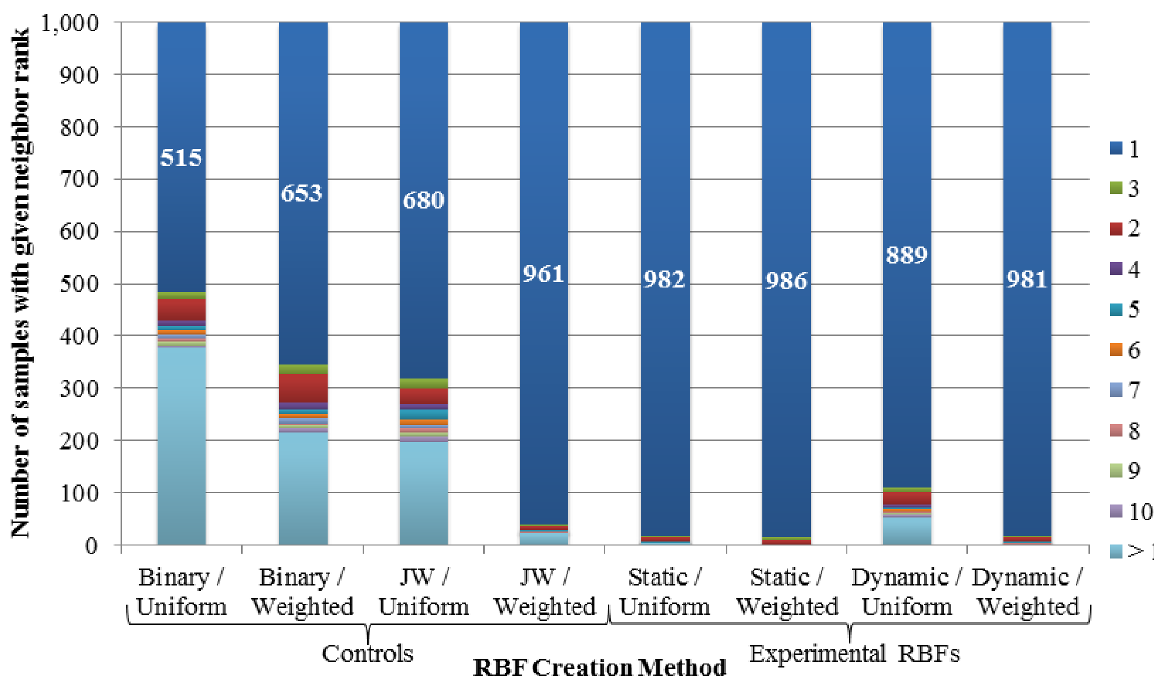


Figure 4.9: The results of the neighborhood test where each color indicates the neighbor rank of the sample's *True Match*. The controls include binary field comparison and the JW string comparator. The experimental techniques are the RBFs based on statically or dynamically sized FBFs.

the experimental RBFs outperform the controls. This indicates that RBFs are more robust in accurately measuring similarity in the face of the types of errors present in this dataset. When comparing the results based on the FBF sizing method, we see that RBFs composed of *dynamic*ally-sized FBFs perform nearly as well as the RBFs composed of *static*ally-sized FBFs.

In all cases, *weighted* bit sampling from FBFs, in accordance with the discriminatory power of the field, performs better than *uniform* bit sampling. Therefore, we recommend *weighted* bit sampling in RBF creation and explore this variant in further experiments below.

Recall that in determining m_{RBF}, we make the assumption that all bits in each FBF should be included in the RBF. We now test whether this assumption is necessary. Figure 4.10 presents the results of the Neighborhood test on condensed RBFs. These results indicate that as m_{RBF} decreases, the extent to which RBF encodings are able to accurately

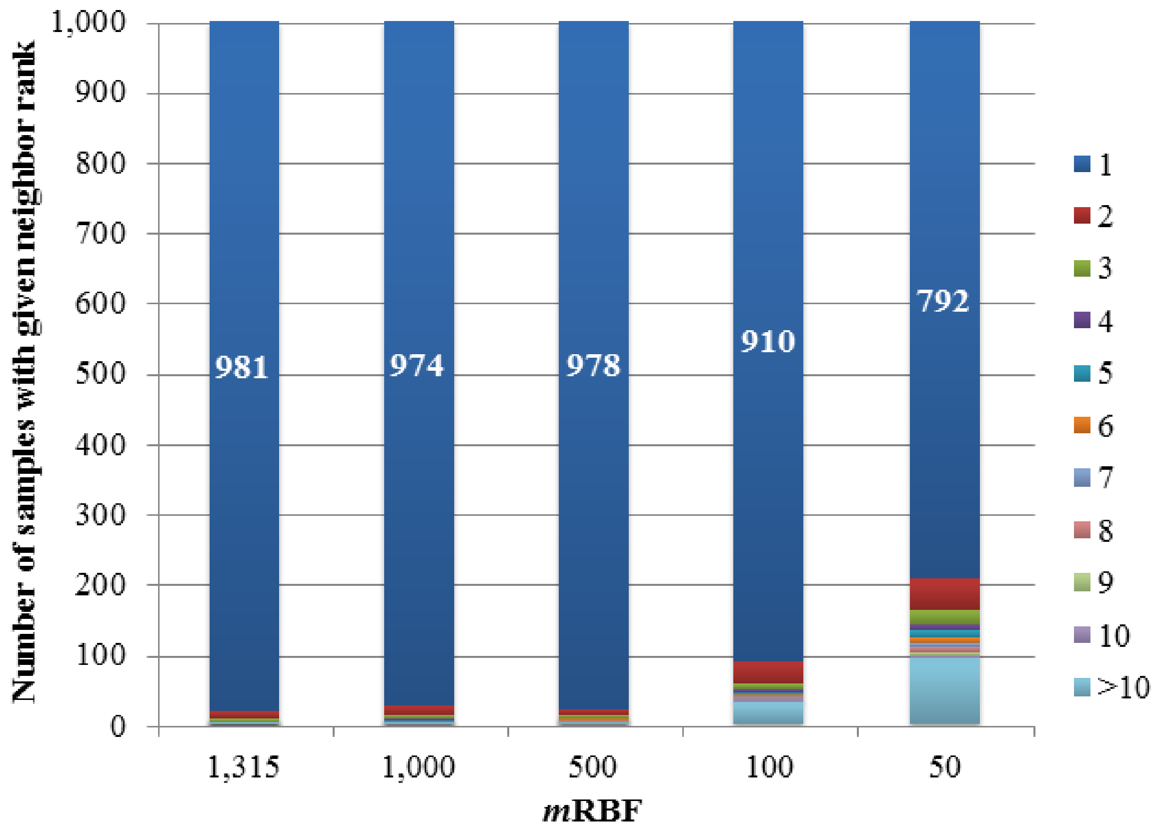


Figure 4.10: The results of the neighborhood test for *dynamic/weighted* RBFs where each color indicates the neighbor rank of the sample's *True Match*. m_{RBF} is indicated for each data series.

measure similarity decreases as well. For example when m_{RBF} = 1,315, the number of samples having a neighbor rank of 1 is 981. When m_{RBF} = 50, only 792 samples have a neighbor rank of 1. However, it does not seem to be vital that the RBF should be sufficiently long to accommodate all bits from all FBFs as RBFs of length less than 1,315 bits still provide nearly as accurate results. For example when m_{RBF} = 1,000 and 500, the number of samples having a neighbor rank of 1 are 974 and 978, respectively.

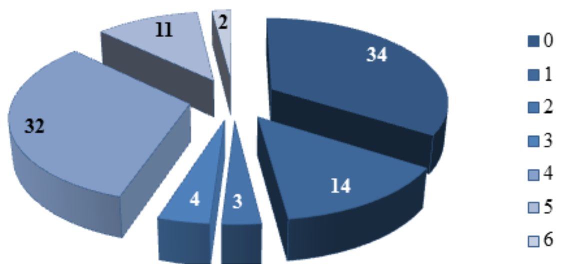
4.4.3 Leveraging Frequency Information to Provide Security Guarantees

As mentioned previously, when an attacker tries to determine the plaintext values encoded in a RBF, a first step may be to map RBF bits back to the fields from which they were drawn, based on the frequency with which a bit is set. For example, if P(bit set) = 0, where P indicates probability, one may conclude that the bit was likely drawn from either the FBF associated with *Ethnicity* or the FBF associated with *Gender*. This is because, as the bit frequency heat maps in Figures 4.7 and 4.8 indicate, *Ethnicity* and *Gender* are the only FBFs containing bits with P(bit set) = 0.

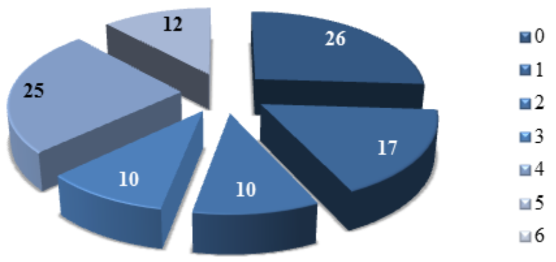
By breaking down the frequencies into ranges, an attacker may use the ranges found in FBFs to map bits in the RBF back to the FBF from which they were drawn. In this analysis, we consider frequency ranges in increments of 0.01. Figure 4.11 shows the number of frequency ranges that be mapped back to q fields where $q \in 1, \ldots, 6$. Figure 4.11(b) indicates that 26 frequency ranges are not represented in any FBFs and are therefore, obviously, not available for inclusion in the RBF. 17 frequency ranges are represented by bits in only a single field. A secure RBF encoding would avoid including these bits because an attacker could easily map these backs to the FBFs from which they were drawn. The extent to which these frequency ranges allow an attacker to map bits back to a FBF is further examined in Figure 4.12, which presents the number and identity of FBFs containing bits in each frequency range, broken down by hundredths.

We can use this information to quantify the extent to which an attacker may be able to map bits back to the FBFs from which they were drawn. However, we can also proactively incorporate this information into a bit selection algorithm for RBF creation to provide security guarantees about the ease with which an attacker may be able to map RBF bits back to fields. It is obvious that bits which can only be mapped back to a single field, such as bits with frequency range (0.78 - 0.79) in *dynamic* FBFs should be avoided, since these bits can only have been drawn from the field *City* (see Figure 4.12(b)). Data holders may want to introduce more stringent security constraints, such as "Only bits that can be mapped, by frequency range, back to $\geq q$ fields can be included in the RBF".

Figures 4.13 and 4.14 demonstrate the relationship between the security constraint q and the number of bits that are eligible for RBF inclusion under that constraint. For example, consider the FBFs of *dynamic* size shown in Figure 4.14(b). If $q = 6$ (*i.e.*, only



(a) *Static* FBFs.

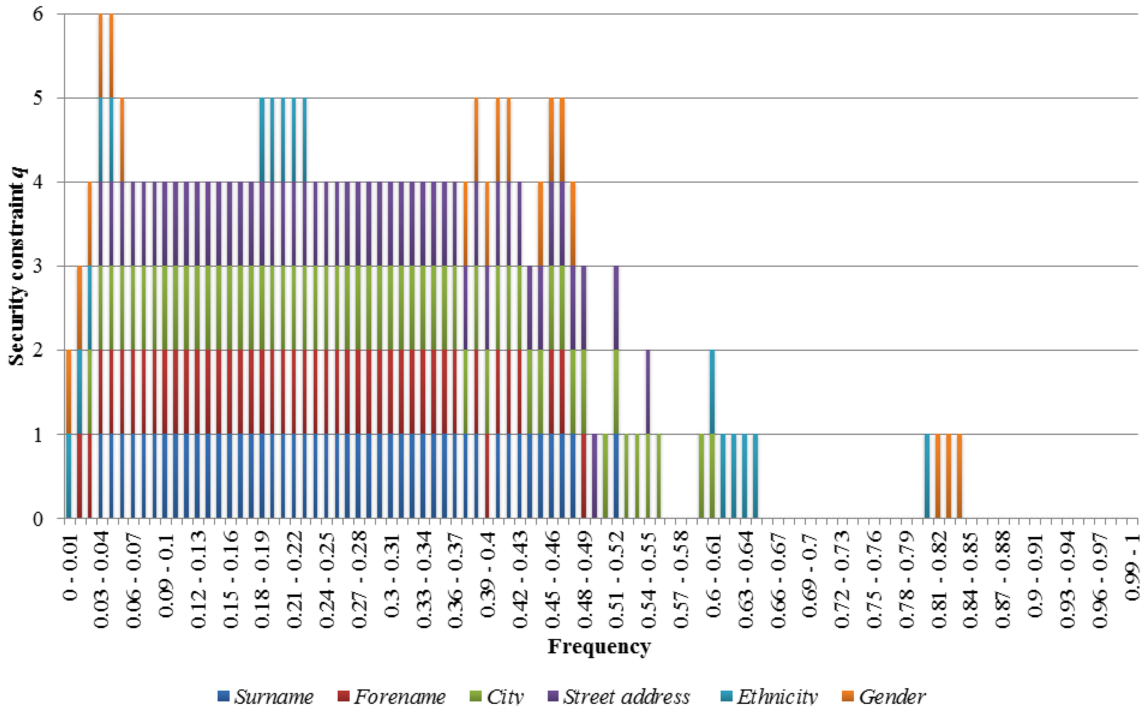


(b) *Dynamic* FBFs.

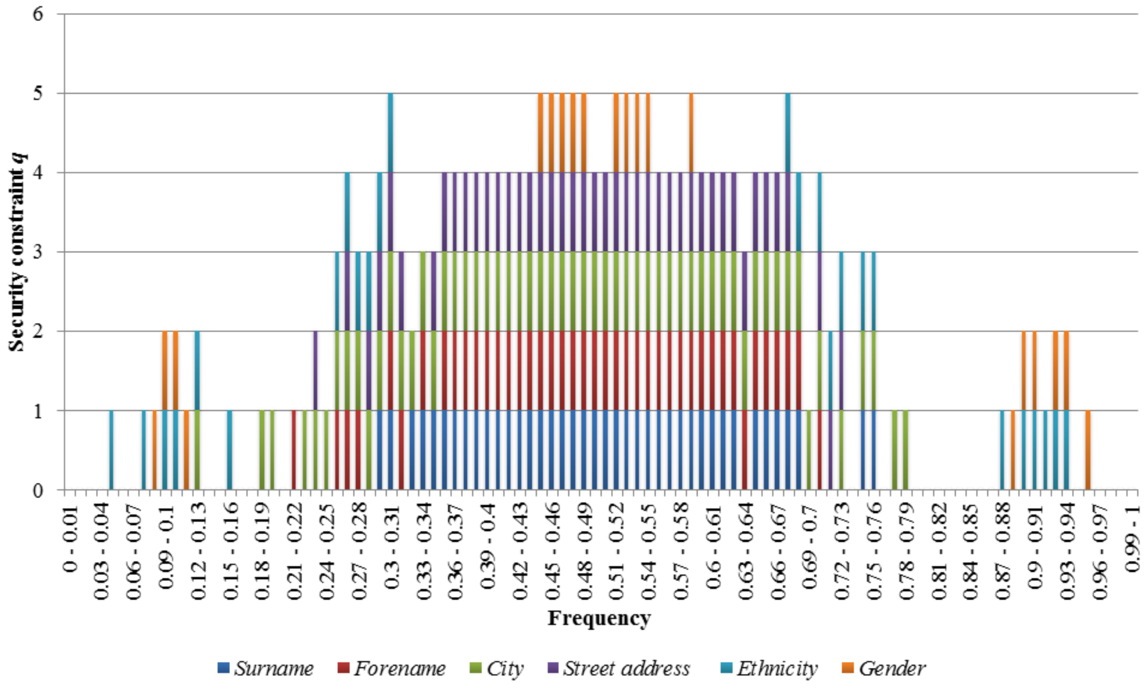
Figure 4.11: Frequency ranges broken down by q, the number of fields containing bits in the specified frequency range.

bits can be included in the RBF that can be mapped back to ≥ 6 fields), no bits that would meet that security requirement. However, if a slightly more relaxed security constraint is used such that $q = 5$, then 312 bits are eligible for inclusion in the RBF.

The relationship between the security constraint q and the number of bits eligible for inclusion in the RBF implies there may also be a relationship between q and accuracy. To examine this relationship, we perform the same Neighborhood test described in Section 4.3.3 on RBFs generated under various security constraints. These results are shown in Figure 4.15. The results in Figure 4.15 indicate that accuracy decreases very slightly (981 samples have a neighbor rank of 1 when $q = 1$, and 970 samples have a neighbor rank of 1 when $q = 5$) when more stringent security constraints are introduced. However, the relationship seems to not significantly affect accuracy and is also not linear. To further explore this relationship, the affect of security constraints is examined in the context of condensed RBFs. These results are shown in Figure 4.16. Again, the results shown in Figure 4.16 indicate that, within a given m_{RBF} value, as q increases, accuracy tends to decrease, as indicated by the decreased number of samples having a neighbor rank of 1,

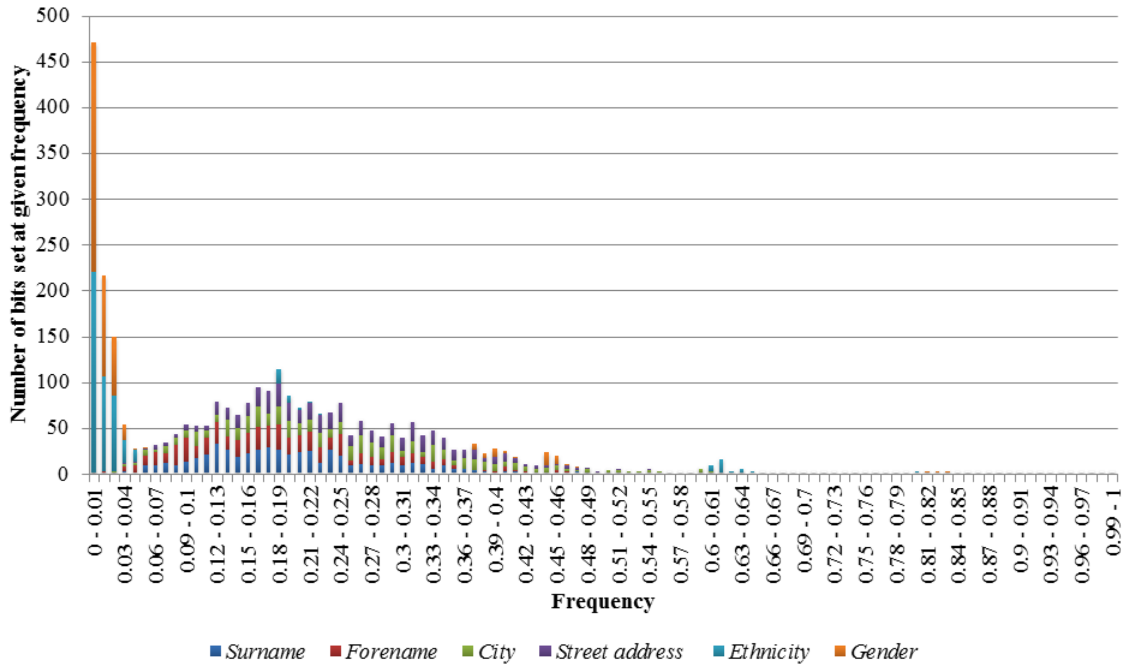


(a) *Static* FBFs.

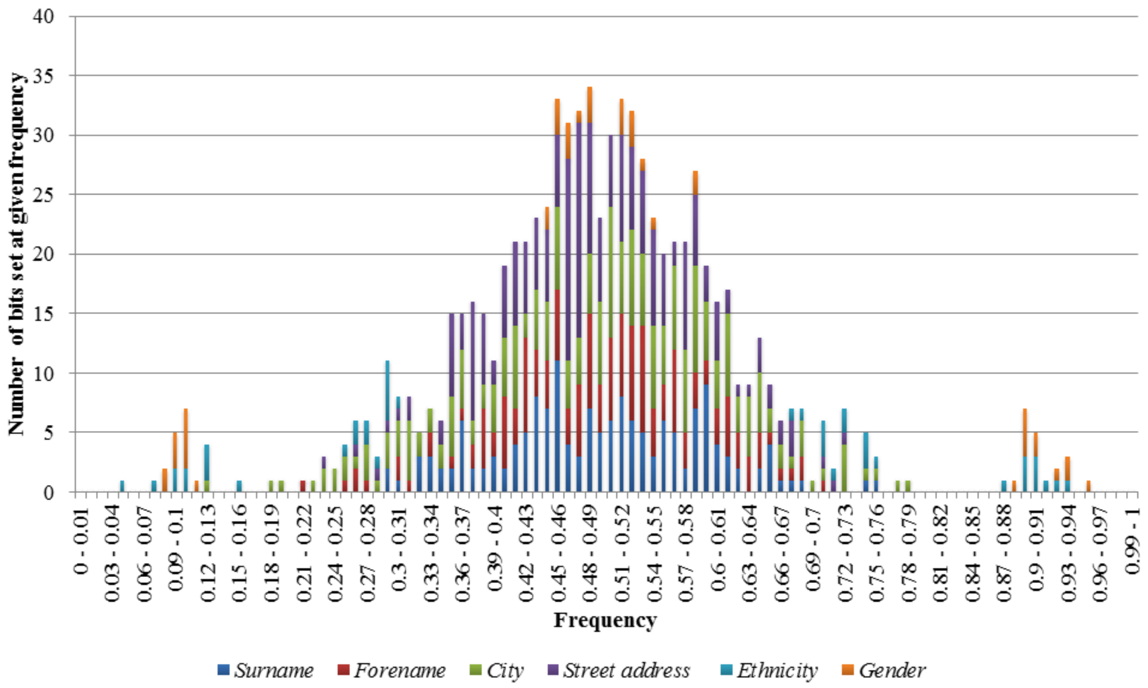


(b) *Dynamic* FBFs.

Figure 4.12: Field frequency histograms show the number of FBFs containing bits set at frequencies in the given ranges.

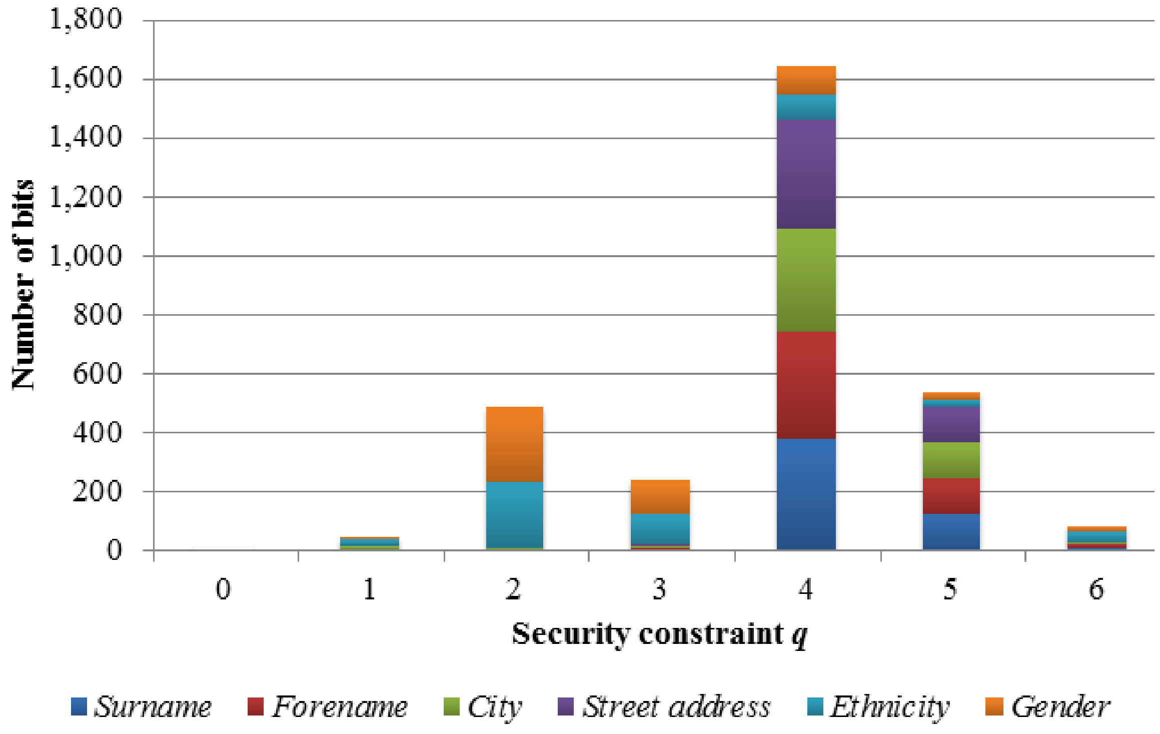


(a) *Static* FBFs.

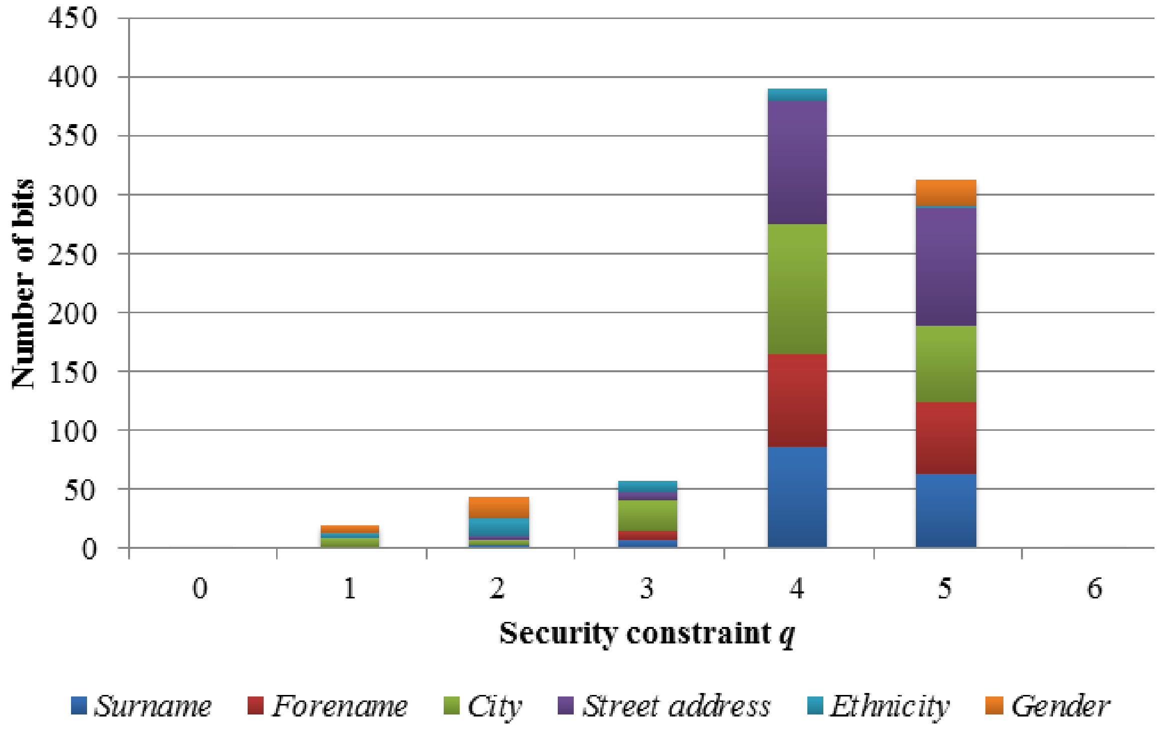


(b) *Dynamic* FBFs.

Figure 4.13: Bit frequency histograms show the number of bits, and the FBFs in which they can be found, set at frequencies in the given ranges.



(a) *Static* FBFs.



(b) *Dynamic* FBFs.

Figure 4.14: The number of bits eligible for RBF inclusion, given various values for the security constraint q.

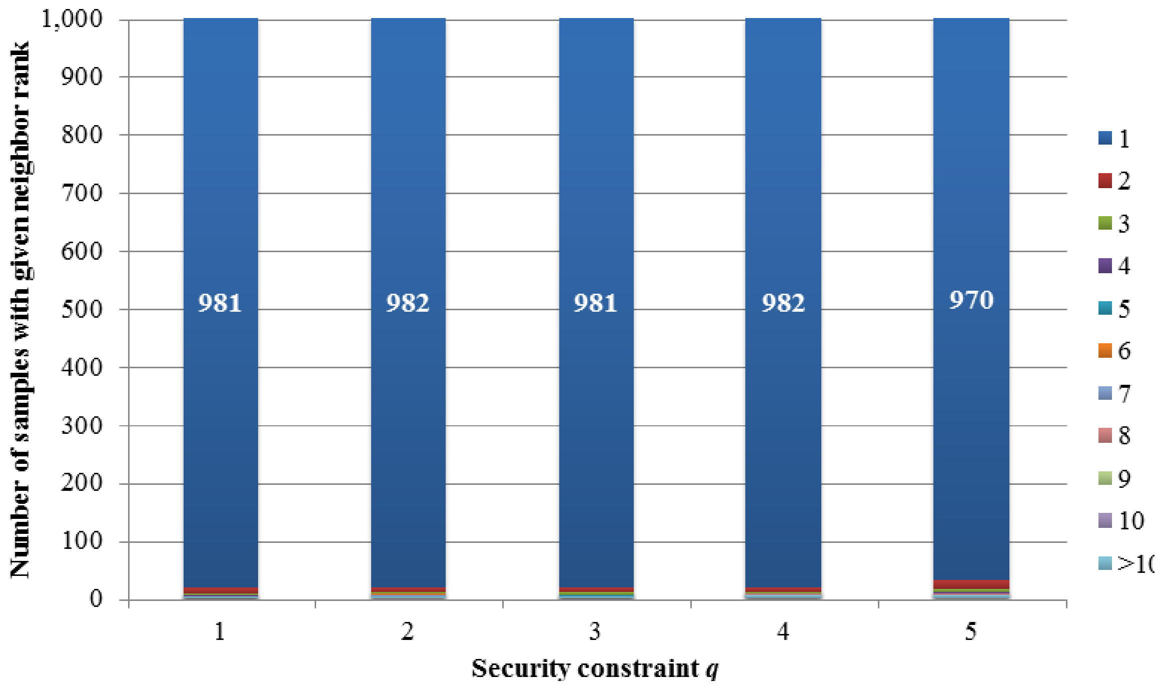


Figure 4.15: Neighborhood test results show that the effect of the security constraint q on accuracy, as determined by the Neighborhood test. Each color indicates the neighbor rank of the sample's *True Match*. The upper horizontal axis refers to the value of the security constraint q. The lower horizontal axis refers to the value of m_{RBF}. The RBFs used are *Dynamic/Weighted*.

although again, the relationship is not linear.

These results are broken down further in Figure 4.17. In this figure, the data series where neighbor rank is 1 (shown in blue in Figure 4.16) is examined in greater detail to model the tradeoffs between q, m_{RBF} and accuracy.The results shown in Figure 4.17(a) indicate that q does not significantly affect accuracy until very small RBF sizes ($m_{\text{RBF}} \leq 100$) are used. Figure 4.17(b) shows that m_{RBF} more significantly affects accuracy – as m_{RBF} decreases, accuracy also decreases – than the security constraint q.

4.4.4 Summary of Results

In Section 4.4.1, we examined the frequencies with which bits are set in both FBFs of *static* and *dynamic* size. The average frequency with which a bit is set differs markedly from field to field within FBFs of *static* size. This is in contrast to FBFs of *dynamic* size, where the average frequency with which a bit is set is approximately the same across fields.

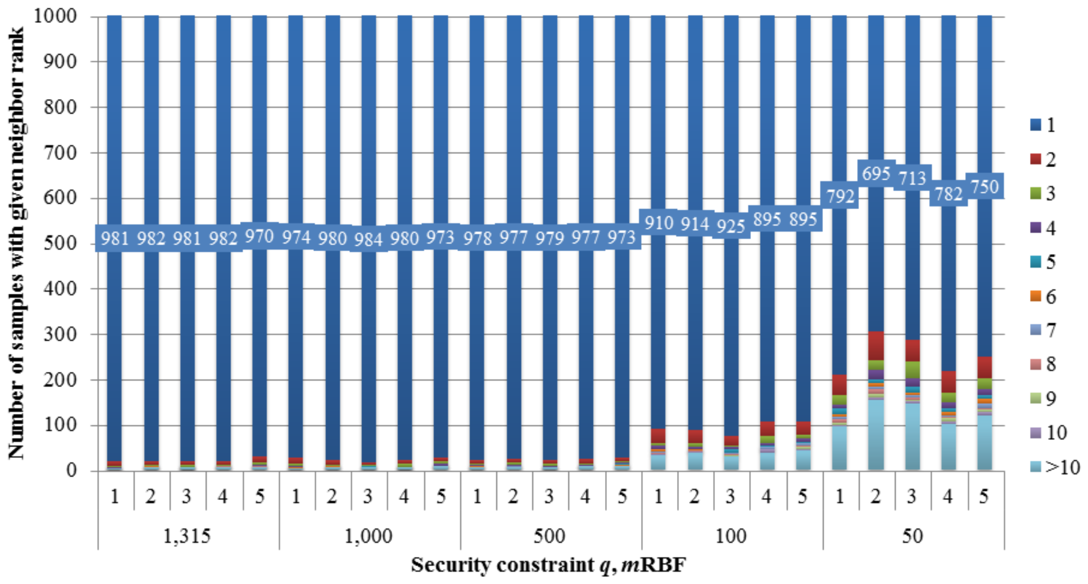
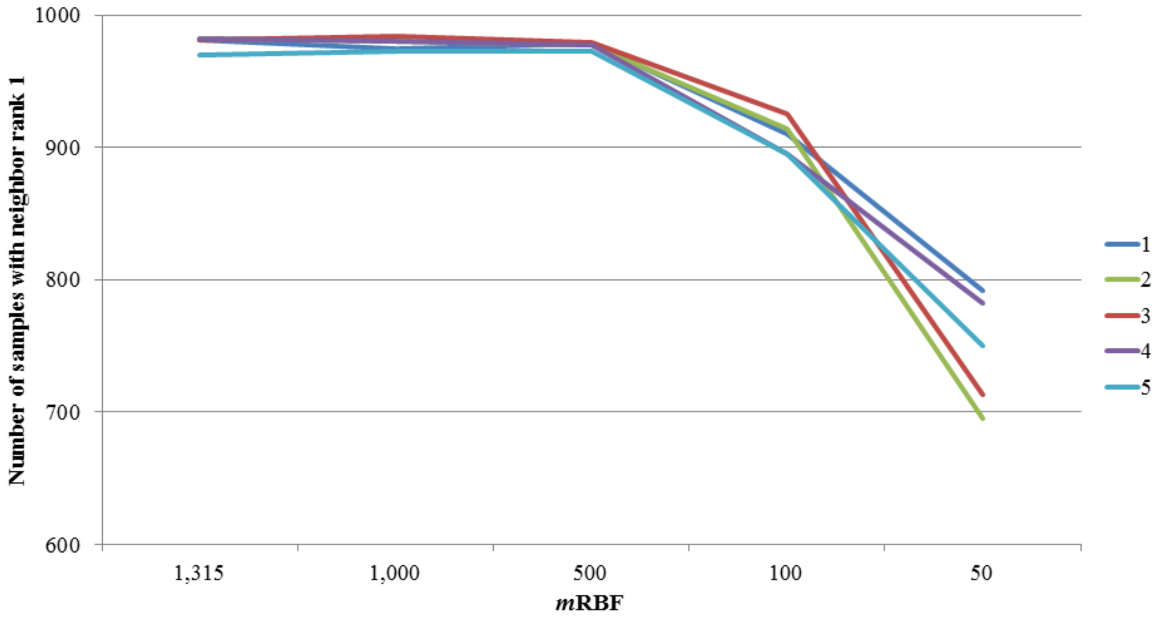


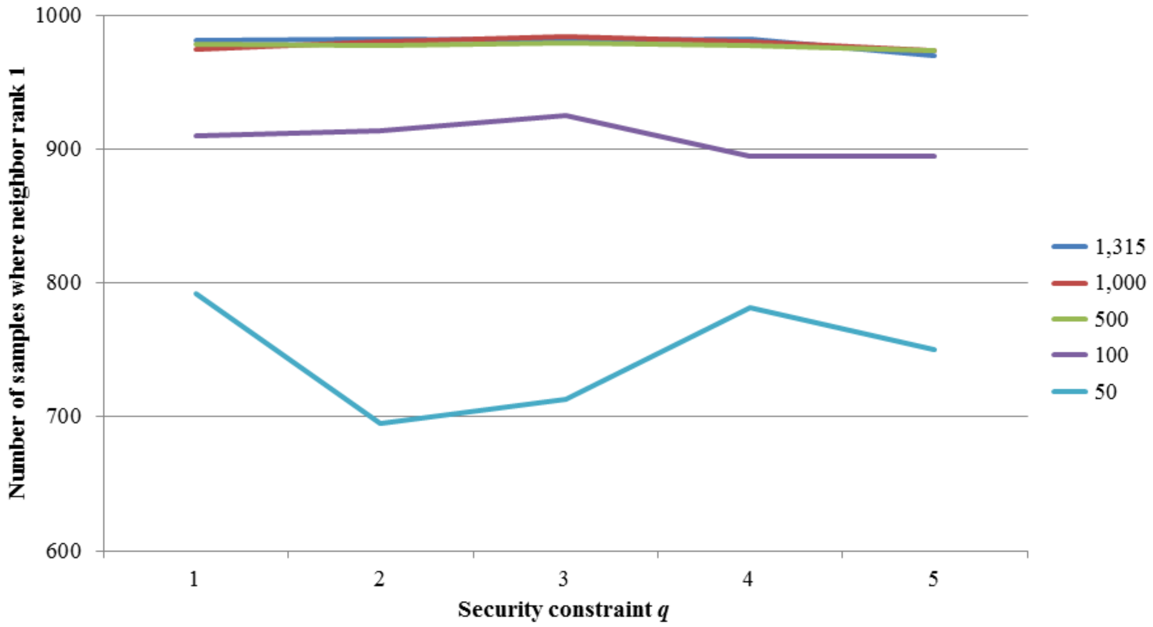
Figure 4.16: These neighborhood test results show the effect of RBF sizing and the security constraint q. Each color indicates the neighbor rank of the sample's *True Match*. The RBFs used are *Dynamic/Weighted*.

Section 4.4.2 showed that Bloom filter-based encodings provide for more accurate record pair similarity detection than several control measures. Additionally, *weighted* bit sampling in RBF generation provides higher accuracy than *uniform* bit sampling across FBFs. Also, we saw that as the RBF is condensed (*i.e.*, the m_{RBF} value is lowered), the accuracy decreases.

Finally, in Section 4.4.3, we demonstrate how an examination of bit frequency ranges, and the number of fields that contain bits in each frequency range, can provide a method for constructing RBFs that are more resistant to attack. Specifically, users can specify constraints, such as "Only bits that can be mapped back to $\geq q$ fields, by examination of frequency ranges, can be included in the RBF." As q grows, the more difficult it is for an attacker to map bits from a RBF back to the field from which they were drawn. RBFs created under this security constraint are only slightly less accurate than those created without any constraints.



(a) Breakdown by m_{RBF}.



(b) Breakdown by security constraint q.

Figure 4.17: A detailed view of the effect of RBF sizing and security constraints on accuracy.

4.5 Discussion

The goal of the work described in this chapter is to develop an encoding for use in PRL that has two properties:

1. **Secure:** The encoding should not compromise the encoded plaintext information.
2. **Distance-preserving:** Records that are similar in the plaintext space should remain similar to one another in the ciphertext space.

In the remainder of this section, we demonstrate that the RBF encoding method satisfies these properties. In Section 4.5.1, we show that RBFs are resistant to the known attack described in Section 4.2.2. In Section 4.5.2, we look at the design decisions involved in RBF generation and how these decisions affect the security and comparison accuracy of the resultant encodings.

4.5.1 Record-level Bloom Filters are Resistant to a Known Attack

In Section 4.2.2, we describe a known attack on FBF encodings based on an analysis of the frequencies at which bits in the FBF are set. The attack is modeled as a constraint satisfaction problem where the task is to map the bits in the encoding back to the plaintext n-grams to which they correspond. This is done by leveraging the frequency with which the bits are set and the frequency with which n-gram values are seen in the general population.

Some preliminary information is needed for this attack. Below we walk through the information required for the attack, and explain why this information is not available, or significantly more difficult to obtain, in the context of RBF encodings.

1. **An attacker must be able to calculate the frequency at which n-grams are present in the general population.** An attacker can use publicly available demographic information, such as census records, to estimate these frequencies. However, when field values are grouped together in a disordered way (*e.g.*, in a permuted RBF), this becomes far more difficult. For example, the frequency distribution of

Forename may be available and relatively distinct. However, the frequency distribution of {*Surname, Forename, City, Street address, Ethnicity, Gender*} is more difficult to calculate and less distinct. Additionally, an attacker must determine which bits in the RBF correspond to each field in order to apply this attack. As discussed in Section 4.4.3, security constraints can be put in place that make this very difficult.

2. **An attacker must know k, the number of hash functions used in encoding.** When FBFs of *static* size are used, an attacker can observe the percentage of bits set on average to estimate k. However, when FBFs of *dynamic* size are used, the attacker loses the ability to estimate k based on observation of bits set.

In addition to not being able to gather the background information needed for this attack, the computational complexity of the attack grows quadratically when field values are grouped together into a single RBF. Previously, an attacker could handle each field individually. Therefore, in order to map the FBFs corresponding to the six fields used throughout this Chapter back to their plaintext value, in the worst case, the attacker would have to perform $|Surname| + |Forename| + |City| + |Street\ address| + |Ethnicity| + |Gender|$ operations. Consider as an example the values shown for the NCVR dataset in Table 2.2. In this case, an attacker would have to perform 388,718 operations. However, when these fields are encoded in the same data structure, an attacker must perform $|Surname| \times |Forename| \times |City| \times |Street\ address| \times |Ethnicity| \times |Gender|$ operations. Using the NCVR dataset as an example again, in this case, an attacker would have to perform 2.97×10^{19} operations! This number of computations is outside the scope of current computing power.

As the attacker cannot gather the background information necessary to mount this kind of attack, and the computational complexity renders it infeasible, the RBF encodings presented in this chapter are resistant to this attack.

4.5.2 RBF Design Decisions

In this section, we discuss design decisions encountered in RBF generation and how these decisions affect the security of the resultant encodings, as well as the speed and accuracy with which they can be compared with one another.

Field selection

The field values encoded can affect the extent to which a frequency attack yields information about the field values. For example, consider the categorical variables *Gender* and *Ethnicity*. The bit frequency distributions of the FBFs corresponding to these fields appear to be dramatically different from the FBFs corresponding to string-based fields, as shown in Figures 4.7 and 4.8. Based on the evidence, we can make a few observations about how the schema affects the resultant security.

The first observation is that categorical variables produce distinct frequency distributions, and are therefore more subject to information leakage as a result of frequency analysis. However, these fields (*e.g.*, *Ethnicity* and *Gender*) also tend to have the least discriminatory power in resolving identity. The second observation is that when fields are drawn from the same domain (*e.g.*, string-valued fields), the less distinguishable they will be in the resultant RBF. Therefore, it will be more difficult to map the RBF bits back to the FBFs from which they were sampled when fields are drawn from the same domain.

Determining $m_{\mathbf{FBF}}$

The next decision in RBF creation is how to determine what should be the value for m_{FBF}. Two alternatives have been put forward and examined in this work. The first is to use FBFs of *static* size regardless of the values to be encoded, and the second is to *dynamic*ally determine FBF size based on the expected length of the field values to be encoded.

The FBFs of *dynamic* size offer several benefits. They prevent an attacker from determining the k value, which is required for the CSP attack described in Section 4.2.2. When FBFs of *dynamic* sized are used, the expected number of bits set when g n-grams are encoded can be controlled by varying the m value, as described in Section 4.3.1. In this work, we let p, the expected percentage of bits set, equal 0.5 in order to maximize entropy. The results shown in Figure 4.6(b) confirm that this is the average percentage of bits set across all FBFs of *dynamic* size, regardless of field type. Thus, no information is revealed by the percentage of bits set in the FBF. This is in contrast to the FBFs of *static* size, shown in Figure 4.6(a), where the average percentage of bits set is distinct across the fields. Therefore, an attacker who seeks to estimate the k value by observing the percentage of bits set, would gain no information about the k value when FBFs of *dynamic* size are used because the average percentage of bits set is always 0.5.

Another property of the *dynamic versus static* sizing of FBFs relates to the signal-to-noise ratio. While the same signal is encoded in each, as indicated by the very similar accuracy measures in Figure 4.9, the amount of noise is far greater in the FBFs of *static* size. In this case, the "noise" corresponds to the many bits that remain at their initial value of 0 in the FBFs of *static* size. Minimizing the noise, while retaining the signal, is desirable for several reasons. Firstly, when more bits are retained in a Bloom filter encoding, it provides an attacker with more inputs to a frequency analysis, which he can then exploit to aid in inferring information about the encoded values. Secondly, bits will be compared in PRL to estimate the similarity of the record pair. Including the extraneous, "noisy" bits in these calculations increases the runtime of an already computationally intensive process. Therefore, eliminating the noise retained in FBFs of *static* size by *dynamically* sizing the FBFs is desirable.

How many **bits from each field to include in the RBF**

Bits can be sampled either *uniformly* across all fields, or according to a *weighting* determined by the discriminatory power of each field. The *weighted* technique seems more intuitive, given the knowledge that different fields are useful at resolving identity to varying extents. Therefore, RBFs that leverage this variation in discriminatory power by sampling more heavily from more informative fields are expected to provide for a more accurate measure of record pair similarity. The results of the Neighborhood test, shown in Figure 4.9, confirm this expectation. In all cases, RBFs created by a *weighted* sampling of bits from FBFs, facilitate a more accurate measure of record pair similarity than RBFs created by a *uniform* sampling of bits from FBFs.

Which **bits to include in the RBF**

Now that we have determined that bits should be sampled from FBFs commensurate with the discriminatory power of the field, the question of *which* bits should be sampled is yet unanswered. We considered two strategies for bit sampling. The first is to randomly sample bits from FBFs and the second is to sample bits from the FBFs using an informed strategy.

For several reasons, we believe that random sampling of bits is the better option. Firstly, use of a more informed strategy would provide a "weakness" an attacker could exploit. If an attacker determined the method used in bit sampling, this may enable him to map bits in the RBF back to the field from which they were drawn more effectively. Secondly, an informed strategy is difficult to develop due to the highly correlated nature of the data structure. As k bits are used to hash each n-gram into the FBF encoding, these bits are highly correlated. Therefore, any informed sampling strategy would have to account for the conditional dependency of bits, which would require additional time. Since one of the benefits of Bloom filter based encodings is the speed with which they can be generated and compared, employing a computationally costly bit sampling strategy would conflict with the goal of a rapid encoding mechanism.

Determining $m_{\mathbf{RBF}}$

Another decision required in RBF generation is determining m_{RBF}, the number of bits to compose the RBF. We test several methods for RBF sizing, including allowing for the inclusion of all bits from FBFs, or setting an arbitary size for the RBF. The results are shown in Figure 4.10. As the results indicate, it is not necessary for all bits in FBFs to be included in the RBF. However, as the size of the RBF is made smaller, the accuracy of record pair comparisons decreases as well.

There are also implications of m_{RBF} for the security of the resultant encodings and for the speed with which the encodings can be compared. As m_{RBF} grows, an attacker has more bits he can input to a frequency analysis. Additionally, the number of bits comparisons required to compare encodings increases, resulting in longer computation time.

In summary, the size of the RBF presents a tradeoff between three desirable properties of encoding mechanisms – security, accuracy, and the speed with with they can be compared. As m_{RBF} increases, security decreases, accuracy increases, and the speed with which they can be compared increases. As m_{RBF} decreases, security increases, accuracy decreases, and the speed with which encodings can be compared decreases.

Determining security constraints

The final decision encountered in RBF generation is what security constraint, q, to enforce. Recall that each bit in the RBF is drawn from a FBF. An attacker may try to map bits in the RBF back to the field from which they were drawn. He observes the frequencies with which bits are set in RBFs and his knowledge of the frequency with which bits are expected to be set in RBFs. By avoiding bits set at frequencies that are unique to a single field, or a small number of fields, the resultant RBF encodings are more secure as it becomes more difficult for an attacker to leverage frequency information to map RBF bits back to the FBFs from which they were drawn. A user can establish a security constraint, q, by allowing only bits that can be mapped back to $\geq q$ fields, by leveraging frequency information, to be included in the RBF. Therefore, the value for this parameter can be determined in each application to be consistent with the desired level of security.

Figures 4.15, 4.16, and 4.17 indicate that enforcing a strong security constraint has little effect the accuracy of similarity comparisons on the resultant encodings.

4.6 Conclusions

We presented a method to make a RBF encoding for use in PRL. Several design choices can be made by users to best suit their requirements with respect to security, the speed at which the encodings can be compared to one another, and the desired accuracy of similarity comparisons based on the encodings. However, our results have shown that using FBFs of *dynamic* size and sampling random bits from FBFs in a measure commensurate with the discriminatory power of the field are important for secure RBF encodings that facilitate accurate similarity calculations. These encodings are more resistant to an attack that has been successful against previously proposed forms of FBF encodings, and therefore provides an encoding for use in PRL that is more secure, yet still facilitates highly accurate similarity comparisons of the encodings.

4.6.1 Future Work

We now describe work left to future research. Preliminary work has shown that RBF encodings are suitable data structures to serve as inputs for blocking methodology in the record linkage process. We explore this further in Chapter 5. Additionally, we believe that the similarity of RBFs are an accurate measure of record pair similarity and can be used directly in PRL to replace the record pair comparison step described in Section 2.1.7. Finally, we plan to investigate how to handle missing field values in the context of Bloom filter encodings. We test these ideas in Chapter 6.

CHAPTER V

PRIVATE BLOCKING

In this chapter, we propose a method for private blocking (PB) based on the RBF encodings presented in Chapter 4. We demonstrate that this method significantly reduces the computational complexity of PRL while maintaining accuracy.

5.1 Introduction

Record linkage, the task of identifying records that refer to the same individual, can be computationally expensive because each record in a dataset must be compared to *each* record in the dataset to which it is being linked. Consider two data providers, Alice and Bob, holding datasets A and B respectively. Suppose |A| = |B| = 100,000. Since each record in dataset A must be compared to each record in dataset B, $|A| \times |B| = 100,000 \times 100,000$ = 10,000,000,000 record pair comparisons are required. In summary, as the number of records input increases linearly, the number of record pairs that must be classified increases quadratically.

Blocking is a technique to reduce the number of record pair comparisons required by removing record pairs from consideration that are unlikely to be *True Matches*. Briefly, blocking is a method to partition records, from datasets A and B, such that the partitions contain records similar to one another. Then, only record pairs within the same partition are compared to one another and classified into the classes *Predicted Match* and *Predicted Non-match*. Since the records not placed in the same partition are not evaluated and classified, they are essentially classified as a *Predicted Non-match* by default. Blocking is further described in Section 2.1.4 and an example of blocking by first letter of *Surname* is shown in Figure 2.2.

While blocking is intended to reduce the number of record pair comparisons required, and thus reduce the computational complexity of record linkage, the specific blocking algorithm and parameters should be chosen carefully since blocking also has the potential to reduce linkage accuracy. Specifically, when record pairs that are *True Matches* fail to be placed into the same partition, they are not compared and are classified as a *Predicted Non-match* by default. Therefore, they are false negatives.

The primary goal of blocking is to reduce the time required to perform record linkage, so it is also important that the blocking algorithm itself does not take long to run. It should be the case that $Time(Blocking) \ll Time(Compare(|A| \times |B|))$.

In summary, an ideal blocking algorithm achieves the following goals:

1. reduces the number of record pair comparisons required,
2. does not reduce the accuracy of record linkage, and
3. runs in significantly less time than that required to compare all record pairs.

5.1.1 Private Blocking

Blocking – in the context of PRL – must satisfy some additional properties to ensure privacy is maintained. Specifically, the blocking algorithm must not require knowledge of the plaintext values, and the properties of the resultant partitions must not reveal information about the originating plaintexts. An example of a blocking algorithm that satisfies the first property, but not the second, is when Alice and Bob agree to send all records where the first letter of the *Surname* is a particular letter (*e.g.*, 'A', then 'B', then 'C' and so on throughout the alphabet), partition by partition, to Charlie for linkage. Though this method does not require revelation of the plaintext values, an attacker who observes these partitions may be able to use their relative size to map each partition back to its associated letter using the frequency information shown in Figure 5.1, generated from the 2000 U.S. Census data [111]. If an attacker is able to attack the data in this manner, it makes the task of mapping encoded values within each partition back to their plaintext values far more

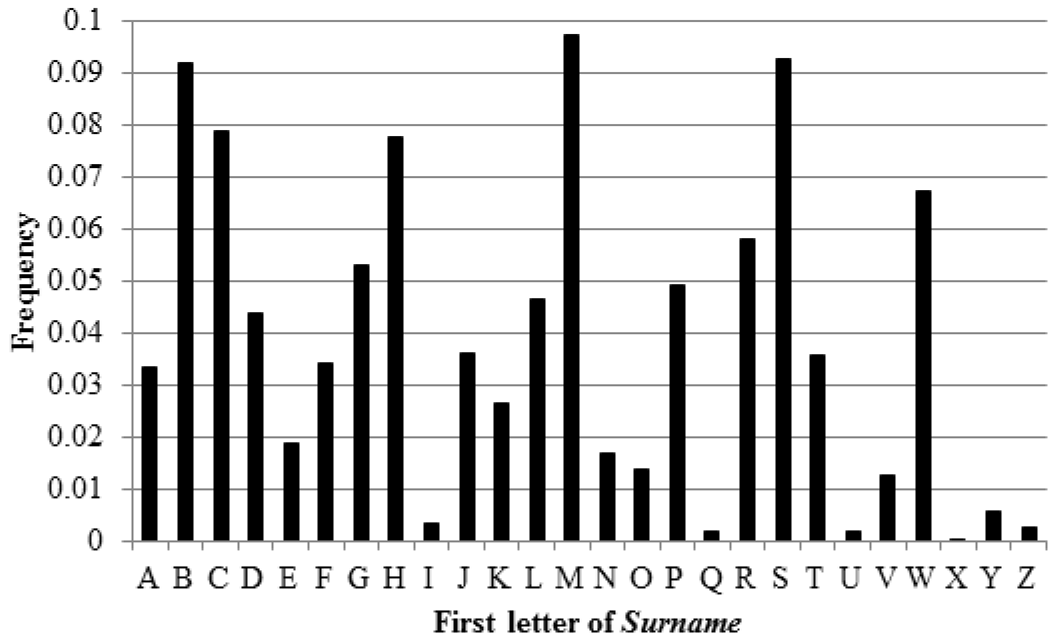


Figure 5.1: The frequency distribution for first letter of *Surname* in the 2000 U.S. Census.

straightforward. Therefore, this protocol does not satisfy the properties of a PB algorithm.

5.1.2 Steps in Blocking

Blocking can be viewed as a function where the inputs are the records from datasets A and B, and the outputs are the set of record pairs to be examined further and classified. Several steps are undertaken in the blocking process, as shown in Figure 5.2. We refer the reader to Section 2.1 and Figure 2.1 to understand how blocking fits into the larger record linkage process.

Further details on the steps involved in blocking are outlined as follows:

1. **Blocking variable identification**

   The blocking variable is the criteria by which the records are partitioned. In the example shown in Figure 2.2, the first letter of *Surname* is applied as the blocking variable.

2. **Spatial partitioning**

   Once a blocking variable has been identified, this variable serves as an input to a

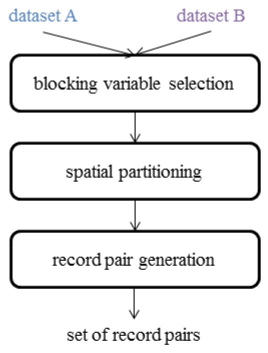


Figure 5.2: The steps in the blocking process.

spatial partitioning process where the output is a set of partitions.

3. **Record pair generation**

   Given a partition of the space, the set of record pairs to be considered further is generated.

In this work, we use the RBF described in Chapter 4 as the blocking variable. An advantage of this data structure is that it includes information from *all* fields in the record. Blocking variables based on a single, or a small number of fields, are more likely to result in false negatives if a field value is not recorded accurately. For example, if first letter of *Surname* is used as a blocking variable, all records where the *Surname* is recorded incorrectly, or where the *Surname* has changed, will be placed in different partitions and therefore incorrectly classified as *Predicted Non-matches*. Incorporating multiple fields in the blocking variable, or iteratively partitioning using different blocking variables, avoids "putting all of one's eggs in one basket" and minimizes the chance of false negatives.

The remainder of this chapter describes a novel method for spatial partitioning of RBFs based on locality-sensitive hashing (LSH). LSH is a class of hash functions where similar inputs are placed in the same bucket with high probability. LSH functions are described further in Section 5.2.4.

5.1.3 Chapter Outline

In Section 5.2, we describe research others have done on blocking in general, and PB in particular. Section 5.3 provides the details of the PB approach proposed in this chapter, the controls against which this new method will be benchmarked, the dataset used in evaluation, and the metrics used to evaluate the methods. Section 5.4 describes the experimental results, and in Section 5.5 their implications, are discussed. Finally, Section 5.6 summarizes this chapter.

5.2 Related Work

This section describes previous work performed on the topic of blocking. Sections 5.2.1 and 5.2.2 describe methods for blocking variable selection and techniques for spatial partitioning, respectively. In Section 5.2.3, we highlight findings of reviews and evaluations of blocking techniques. In Section 5.2.4, we provide details on LSH since it lays the groundwork for the methods proposed in this chapter.

5.2.1 Blocking variable selection

The first step in blocking is to identify a blocking variable which will guide the partitioning process. The blocking variable may be composed of information from a single field, or multiple fields. Some examples of blocking variables are:

- *Surname*
- first letter of *Surname*
- first letter of *Surname* AND *Month of Birth*

Michelson *et al.* present a machine learning approach to identify the best blocking variable or variables [87]. Another work on blocking variable selection [21] takes an analytical approach to finding the best combination of field values, and portions of field values, to comprise the blocking variable. The authors prove that finding a globally optimal solution is NP-hard and present a greedy approximation algorithm.

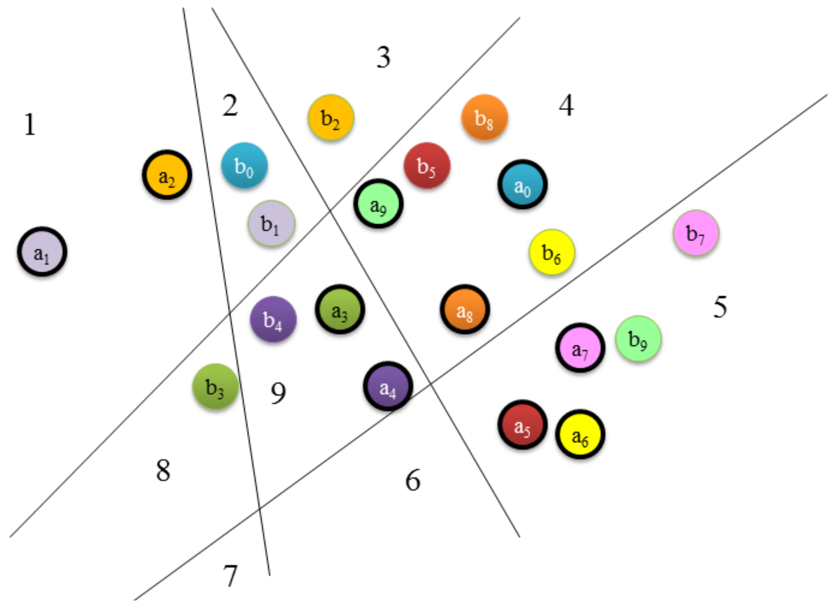
5.2.2 Spatial Partitioning

This section presents previous work on partitioning records, given a blocking variable. The term "spatial partitioning" is particulary relevant when each record is viewed as a single point in a multidimensional space. The goal of this step of the blocking process is to generate a partition of this space, such that similar records are placed in the same partition.

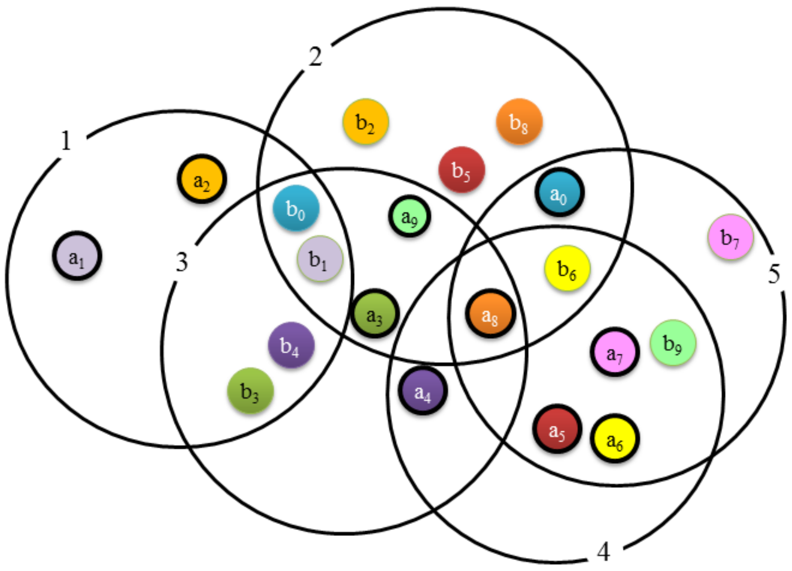
Many of the approaches share properties and are broken into four classes: 1) token-based blocking approaches, 2) sorting-based blocking approaches, 3) n-gram based approaches, and 4) clustering approaches. Throughout the remainder of this section, we describe the properties of these classes and provide examples of each.

Another property of partitions that spans the categories below is that a partition can either be hard or fuzzy. We mention this distinction because the partition type affects how the record pair generation step must be performed. When a hard partition is created, each record is placed in only a single partition, as shown in Figure 5.3(a). By contrast, a fuzzy partitioning method allows each record to be placed in multiple partitions, as shown in Figure 5.3(b).

When a hard partition is created, the record generation step is straightforward because the records from disparate datasets within each partition are compared to one another. Consider the example shown in Figure 5.3(a). The record pair comparisons generated from partition 5 are $\{(a_7, b_9)$, (a_7, b_7), (a_5, b_9), (a_5, b_7), (a_6, b_7), $(a_6, b_9)\}$. Partitions 6 and 7 do not generate any record pair comparisons because they do not contain any records. Partitions 3 and 8 do not generate any record pair comparisons because they only contain a single record, and thus there is no record from the opposing dataset to which it can be compared. Partitions 1 and 2 do not generate any record pair comparisons because they contain records from the same dataset. Since the goal is to link records *across* the two


1
2
3
4
5
6
7
8
9
a_0 a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8 a_9
b_0 b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8 b_9

(a) Hard partitions.


1
2
3
4
5
a_0 a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8 a_9
b_0 b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8 b_9

(b) Fuzzy partitions.

◉ record from file A ● record from file B

Figure 5.3: An example of hard *versus* fuzzy spatial partitions. In hard partitions (5.3(a)), a record is placed in only a single partition. For illustration, each partition is assigned a number. In fuzzy partitions (5.3(b)), a record can be placed in multiple partitions. For illustration, each partition boundary is assigned a number.

datasets, records in the same dataset are not compared to one another.

However, when a fuzzy partitioning method is used, record pair generation is less straightforward. Consider the example shown in Figure 5.3(b). If the set of record pair comparisons generated by each partition boundary is used as input to the subsequent steps of record linkage, some record pair comparisons would be generated multiple times. For example, record pair (a_8, b_6) is included in partitions 2, 4, and 5. However, comparing this record pair more than once in the context of record linkage would be redundant. Therefore a global hash table that does not allow for the inclusion of duplicates should be applied to maintain a unique list of record pair comparisons required for a record linkage protocol.

It is also noteworthy that any hard partitioning method, when applied repeatedly, results in fuzzy partitions, as shown in Figure 5.4.

In the remainder of this section, we discuss specific spatial partitioning approaches. When appropriate, we point out whether each approach creates hard or fuzzy partitions.

Token-based blocking approaches

Token-based blocking approaches create a hard partition by leveraging a token, selected from the field values in the record, to place the record into a partition. The blocking example shown in Figures 2.2 and 2.3, where first letter of *Surname* is used as the blocking variable, is an example of token-based blocking.

Fellegi and Sunter [46] proposed that all records with the same value for the blocking variable are inserted into the same partition. In this case, each record is inserted into only a single block. However, Christen points out that errors in the blocking key result in false negatives and that the sizes of blocks generated depend on the distribution of the blocking variables values [29].

An approach presented by Tejada *et al.* seeks to overcome the false negative rate associated with blocking algorithms that place each record into a single partition by breaking up each field into a set of tokens, which are subsequently used as a blocking variable [107]. For example, if the field *Restaurant name* has the value "Art's Delicatessen", the

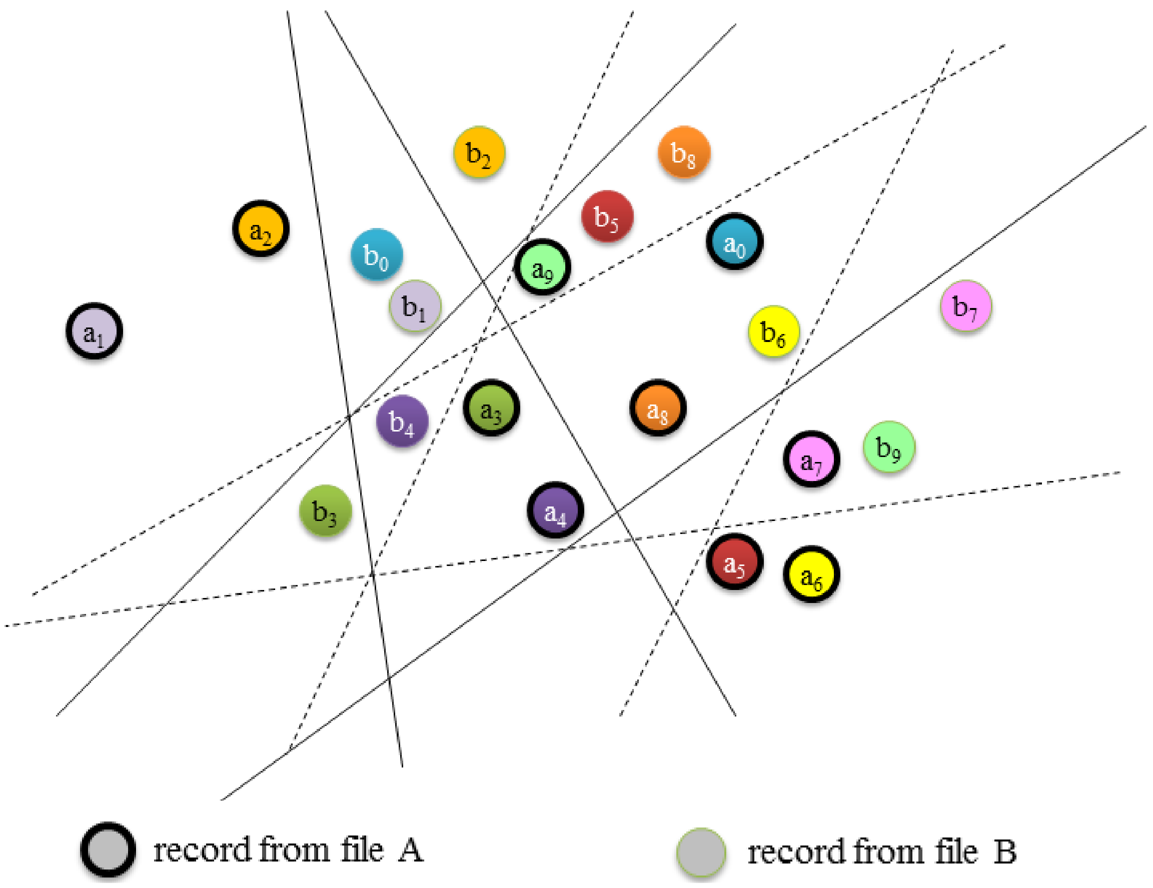


Figure 5.4: Fuzzy partitions can be derived from the repeated application of a hard partitioning strategy. In this example, a second hard partition, shown with dashed lines, is applied.

corresponding tokens are “Art”, “s”, and “Delicatessen”.) Tejada also presents several “transformations” that can be applied to tokens. Valid transformations are stemming – which converts a token to its “root” (i.e., “Delicatessen” is converted to “Deli”), applying a phonetic filter, and abbreviation expansion (*e.g.*, “St.” is converted to “Street”). All record pairs are considered for further evaluation if any token, or transformed token, agrees amongst the record pairs. This approach is *ad hoc* and requires knowledge of the plaintext values and is therefore not suitable for PB.

Al-Lawati *et al.* presents a multi-pronged approach to blocking [10]. In the first step, every pair of records becomes a candidate pair if they share at least one token. In the second step, the binary representations of each record’s hash signature is compared using the Jaccard coefficient, a set-based similarity function. If this value is greater than a predefined threshold, the record pairs are compared.

Another approach is based on using the suffix of the blocking variable to insert a record into each partition corresponding to the values in its suffix array [9]. The suffix array is created by inserting the suffixes of the blocking variable into the array until some minimum length is reached. For example, if the blocking variable value for a given record is “durham” and the minimum length is 3, the suffix array contains the values “durham”, “urham”, “rham”, “ham”. The record is inserted into the four partitions corresponding to the values in the suffix array. This approach also attempts to control partition sizes by setting a maximum partition size. Partitions containing a number of records that exceeds the maximum partition size are considered too general, and the record pair comparisons indicated by these partitions are not considered for further evaluation.

Sorting-based blocking approaches

The blocking algorithms in this category involve sorting all records, by blocking variable, and sliding a “window” along the set of sorted records [54]. All records in the same window frame are placed into the same partition. Therefore, the window size dictates partition size and in how many partitions each record will be placed. One problem with this approach, pointed out in an evaluation [15], is that if the number of records having the same value for the blocking key are larger than the window size, then all records having the same blocking key are not compared.

Monge *et al.* [88] use a similiar approach, but propose a different sorting algorithm that involves sorting in the traditional manner, and a second sorting pass where blocking variable values are read in reverse order.

One challenge associated with applying these techniques in PB is that there is no clear way to sort records in a privacy-preserving way. Additionally, the process of sorting itself can be very expensive. The aforementioned approaches control partition size, as the size of the partitions is dictated by the window size. However, they do not control for the number of partitions, as each record is part of a partition "centered" at that record. As the number of records becomes large, this approach may not guarantee suitable time savings.

n-gram based approaches

This class of approaches involves breaking down the blocking variable values into their associated n-grams to guide the spatial partitioning.

One approach is based on bigram indexing [15] in which the blocking variable value is used to generate a bigram list, a subset of combinations of the bigrams composing the value. A threshold is applied to calculate the length of the bigram lists and thus influence how many partitions each record is placed.

For example, if the blocking variable value is "durham", the set of bigrams associated with this values is "du", "ur", "rh", "ha", "am". If the threshold is 0.8, the bigram lists should be of size $|\textit{set of bigrams}| \times \textit{threshold} = 5 \times 0.8 = 4$. Therefore, all permutations of size 4 of the set of bigrams are generated as follows.

{"du", "ur", "rh", "ha"}

{"du", "ur", "rh", "am"}

{"du", "ur", "ha", "am"}

{“du”, “rh”, “ha”, “am”}

{“ur”, “rh”, “ha”, “am”}

Each record is inserted into the partitions associated with its bigram lists. Therefore, a record where the blocking variable value is “durham” is inserted into five blocks in this example.

One downside of n-gram based blocking algorithms is that the resultant partitions are subject to frequency analysis in the same way demonstrated in Figure 5.1.

Clustering and data partitioning

This set of blocking algorithms views blocking as a data clustering problem where clusters of records are identified and then partitioned.

Ordonez *et al.* propose the use of c-means clustering, an approach where the number of clusters, c, can be determined by the user [82], for partitioning binary data [90]. While this approach initially seems promising for use on binary Bloom filter encodings, we do not believe it is a good fit for partitioning RBF encodings for several reasons. Firstly, this approach assumes that when comparing bit values, a “0/0” agreement (where the values of both bits being compared is 0) is as informative as a “1/1” agreement (where the values of both bits being compared is 1). However, this does not hold for Bloom filter based encodings because “0/0” agreements are not informative in determining the similarity of the encoded values due to the fact that all unset bits are initialized with the value ‘0’. Additionally, it can be difficult for a user to foreknow the most appropriate c value, making this approach difficult to parameterize. Finally, c-means clustering is known to be an NP-hard problem, and therefore heuristic algorithms must be used.

Another blocking approach is based upon binary matrix decomposition [80], but is too computationally intense. Recall that the primary goal of blocking is to speed up the record linkage process; therefore, computationally intensive blocking schemes are undesirable. This approach also requires foreknowledge of the expected number of clusters.

Others have suggested using an r-tree, a data structure for storing and accessing spatial data, for spatial partitioning [59] in conjunction with the *String map* algorithm [67]. However, a comparative evaluation of blocking approaches indicated this approach does not perform well [15].

One of the more popular blocking algorithms, used by several others with success [20, 22, 36, 39], is called the *Canopy* approach [86]. As we use this approach as a control method for the experiments in this chapter, we will provide details about the way it works. This approach is based on the use of a "canopy", the set of records for which the blocking variable value is within some distance from a central point. Users can specify the distance, or similarity, metric used, as well as the threshold. Two threshold values, t_1 and t_2, are required where $t_1 > t_2$. Let S be the set containing all records from datasets A and B. Select a random record r from S, which will serve as the centroid for this canopy, and measure the distance from this record to all other records. All records within a distance of t_1 are placed in this canopy. All records within a distance of t_2 are removed from S. This process is repeated until S is empty. Therefore, an element may appear in more than one canopy, and every element must appear in at least one canopy. This approach differs from the sliding window approach [54] approach in that rather than each record serving as the centroid of a partition, only a subset of the records serve as a partition (*i.e.*, canopy) centroid. The idea is that a cheap distance metric is used in canopy creation and that a more expensive, accurate distance metric is used to evaluate the record pairs that are flagged for further evaluation. However, the computational complexity can still be excessively large since many record pair comparisons may be required (depending on the threshold values used).

5.2.3 Blocking evaluations

Baxter *et al.* performed a review and evaluation of several blocking algorithms [15]. Three performance metrics, first introduced by Elfeky *et al.* [43], are used to evaluate these algorithms. We define them here because we also use these evaluation metrics in this chapter. Let e be the number of record pairs output by a blocking method and let $L = |A| \times |B|$.

To examine the extent to which each blocking algorithm reduces computational complexity, a metric called the *Reduction ratio* (RR) is introduced and defined in Equation 5.1.

$$RR \ = 1 - \frac{e}{L} \tag{5.1}$$

Let e_{tp} be the number of *True Matches* in the set of record pairs produced for comparison by the blocking method and L_{tp} be the total number of *True Matches* in the dataset. *Pairs completeness* (PC), defined in Equation 5.2, is a measure of how well the blocking algorithm retains the record pairs that are *True Matches* for further processing.

$$PC \ = \frac{e_{tp}}{L_{tp}} \tag{5.2}$$

The *F-score* is the harmonic mean of two variables and reflects the extent to which each blocking algorithm balances these desiderata. In this chapter, we examine the *F-score* of PC and RR.

$$F - score(\tau, \upsilon) = \frac{2 \times \tau \times \upsilon}{\tau + \upsilon} \tag{5.3}$$

The results in Baxter *et al.* [15] show that a token-based blocking technique (where the blocking variable is the first four letters of *Surname*) performs approximately as well as the sorted neighborhood method. The n-grams based blocking algorithm described in Section 5.2.2 outperforms both of these methods, but the *Canopy* approach described in Section 5.2.2 performs best overall.

Guidance on blocking variable selection is also provided in the review by Baxter *et al.* [15]. For instance, it was found that less informative variables, such as *Gender*, do not greatly decrease the computational complexity. However, a very informative field, such as

SSN, may decrease accuracy if it is not always recorded correctly. A "middle ground" field, or combination of fields, is desirable. Additionally, use of the least error-prone fields was found to be desirable in a blocking variable.

Elmagarmid *et al.* also present a qualitative discussion of blocking approaches [44] and Yancey notes the difficulties associated with blocking variable creation [121]:

> *"Hence the choice of blocking criteria is something of an art and since any choice may result in overlooking some pairs of likely matched records, a record linkage project usually involves successive runs using different blocking criteria. Recent matching projects have used up to 10 blocking passes."*

5.2.4 Locality Sensitive Hash Functions

We now provide background information on locality sensitive hashing (LSH) generally and how it has been used in blocking. We introduce two locality sensitive hash function families, based on the Jaccard and Hamming distances, respectively. We describe the blocking algorithm based on the Jaccard family of locality sensitive hash functions. We provide background on the Hamming family of locality sensitive hash functions, which provides the basis for the blocking algorithm we propose in this work and is discussed further in Section 5.3.

LSH is a probabilistic method for dimensionality reduction, where a set of hash functions are used to map objects into partitions, such that similar objects share a bucket with high probability, while dissimilar ones do not. LSH has been used for nearest neighbor search [102], genetic sequence comparison [27], and image retrieval [74].

LSH involves the use of locality sensitive hash function families, which are defined as follows. Let r_1 and r_2 be two distances such that $r_1 < r_2$. Let p_1 and p_2 be two probabilities such that $p_1 > p_2$. Then a family of hash functions $G = \{h : F \longmapsto W\}$ is said to be (r_1, r_2, p_1, p_2)-sensitive if for any $f_1, f_2 \in F$ and for any $h \in G$

- if $dist(f_1, f_2) \leq r_1$, then $P[h(f_1) = h(f_2)] \geq p_1$.

- if $dist(f_1, f_2) \geq r_2$, then $P[h(f_1) = h(f_2)] \leq p_1$.

The family of hash functions G can be chosen such that they approximate other functions. In the remainder of this section, we describe hash function families for approximating the Jaccard and Hamming distances, respectively.

Jaccard LSH

$MinHash$ is a class of hash functions that approximate the Jaccard distance and have been used in blocking [72]. We call this approach Jaccard-based locality-sensitive hashing, or JLSH. The Jaccard coefficient is used to measure the similarity of two sets I and J and is defined in Equation 5.4.

$$Jaccard(I, J) = \frac{|I \cap J|}{|I \cup J|} \tag{5.4}$$

The $MinHash$ function accepts binary input, randomly permutes the bits, and scans from left to right until a bit with a value 1 is reached. The index of this bit is the hash value. All records with the same hash value are then mapped to the same partition.

We refer the reader to the work by Broder [26] for a more thorough explanation and proof that $MinHash$ approximates Jaccard distance.

Hamming LSH

Hamming distance is the number of positions at which strings differ and is defined in Equation 5.5.

$$Hamming(I, J) = I \oplus J \tag{5.5}$$

where $\oplus$ indicates the "exclusive or", or XOR, operation.

Given an understanding of Hamming distance, we take a look at the family of hash functions that approximate this distance (HLSH). The applicable hash function is $h_\rho(I) = I[\rho]$. A random ρ is selected, and the value of the ρ^{th} index of I is the hash value. Essentially, this process is the same as the random bit sampling strategy described in Section 4.3.1. We refer the reader to the work by Indyk *et al.* [64] for a more thorough explanation and proof that random bit sampling approximates Hamming distance.

5.3 Materials & Methods

5.3.1 Experimental Methods

We believe that the class of LSH hash functions approximating Hamming distance (HLSH), as described in Section 5.2.4, when applied to RBF encodings, creates a set of partitions useful for blocking. The RBFs are made from FBFs of *dynamic* size by a *weighted* sampling of random bits in a measure commensurate with their discriminatory power. The value of m_{RBF} used in this work is 1,000 bits. Please see Chapter 4 for further details. To summarize, each RBF, containing m bits, can be viewed as a single point in an m-dimensional space, and we propose partitioning this space using HLSH.

We introduce two parameters, ϕ and μ. Let ϕ be the number of individual hash functions applied in the family and let μ be the number of iterations used. Therefore, for HLSH the ϕ randomly selected bits must agree in order for records to be placed into the same partition. Since each bit can have two values, a maximum of 2^{ϕ} hard partitions are created. *True Matches* do not always agree for each bit value in the Bloom filter encoding, so it is possible that the records will fail to agree on all ϕ randomly selected bits. Therefore, this procedure is applied iteratively, and another ϕ random bits are sampled in the next iteration. Although a hard partition is created at each iteration, when $\mu > 1$, a fuzzy partition results, as shown in Figure 5.4. In this case, we use a global hash table in record pair generation, as described in Section 5.2.2.

In experiments, we evaluate HLSH, with multiple parameter settings. Specifically, we consider values for ϕ in $\{14, 16, 18, 20\}$ and values of μ in $\{0, \ldots, 100\}$.

5.3.2 Controls

We use a "token-based" blocking approach where multiple iterations are applied based on the first letter of each field. Therefore, if the first letter of the first field agrees, the record pairs are evaluated; if the first letter of the second field agrees, the record pairs are evaluated; and similarly for any number of fields adopted for the token-based blocking approach. We use the parameter ν to refer to each iteration. We exclude two fields – *Ethnicity* and *Gender* – because their domains are composed of only a few values, such that the resultant partitions would not sufficiently reduce the number of record pair comparisons. Therefore, in this control method, we let $\nu = 4$, and create partitions based on the first letter of the fields *Surname*, *Forename*, *City*, and *Street address*.

We also compare to the $MinHash$ family of LSH functions that approximate Jaccard distance. This class of function has recently been proposed for use in blocking and has shown promising results [72]. The input to this function is the RBF encoding.

Finally, we also compare to the $Canopy$ approach. As mentioned earlier, this approach has been widely used for blocking [20, 22, 36, 39]. This approach requires the user to set several parameters. Specifically, we must define a distance function, a loose threshold, t_1, and a tight threshold, t_2. The specific applications we use are as follows:

- **Data structure:** For the input data structure, we use a subset of the bits in the RBF. The reasoning here is that the calculations required in canopy creation should be cheap and therefore efficient. To facilitate this aspect of the process, we perform calculations over a random subset of the bits in the Bloom filter encoding. The number of bits sampled is called ϕ, in the same way discussed in Section 5.3.1, and we test several ϕ values.

- **Distance measure:** Hamming distance, as defined in Equation 5.5 is used for measuring the distance between RBF encodings. This is selected because it is one of the cheapest, in terms of running time, distance metrics for binary data.

- **Threshold values:** We test several values for t_1 and t_2. The values selected depend upon the ϕ value selected in input data structure creation.

5.3.3 Evaluation Metrics

We use several metrics introduced by Elfeky *et al.* [43] to evaluate the accuracy and reduction in computational complexity afforded by each blocking algorithm. These metrics are RR, PC, and the *F-score* of these values, and are defined in Equations 5.1, 5.2, and 5.3, respectively. To examine the computational complexity of the blocking algorithm itself, we record the running time required for algorithm execution.

5.3.4 Dataset

We use 100,000 records from the NCVR datase, and a second set of records, to which the original is linked, is generated using a data corrupter, as described in Section 2.4.1. This dataset was also used for the experiments presented in Chapter 4.

We exclude records with missing field values so that we get a clear idea of the performance of each method without this complicating factor. We address handling missing values in the context of Bloom filter encodings in Section 6.3.1.

5.3.5 Implementation Details

As noted in Section 5.2.2, a global hash table is required for record pair generation when fuzzy partitions are created. The use of a global hash table requires that the number of record pair comparisons required be of sufficiently small size that they can be stored in system memory. If, however, the number of record pair comparisons required exceeds system memory, an alternate method is required such that the output record pairs are written to disk in such a way that duplicate values are not allowed. To accomplish this, we use a MySQL database for record pair generation where the record pair ID serves as the primary key to ensure that duplicate record pairs are not allowed. When the number

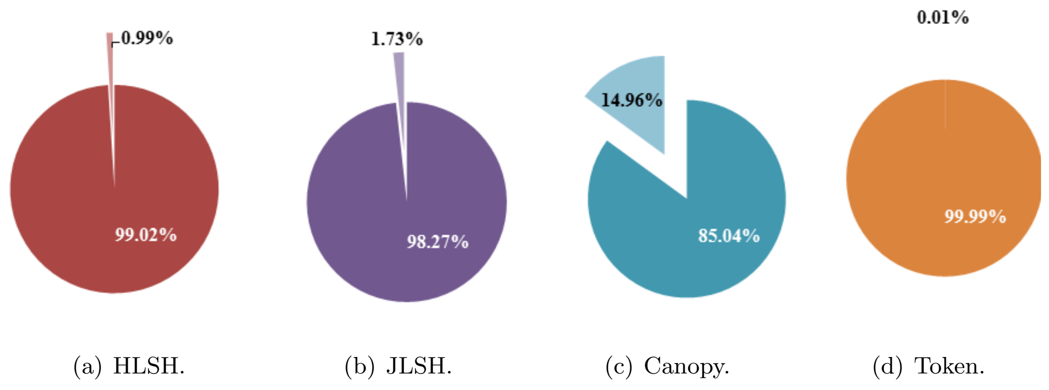


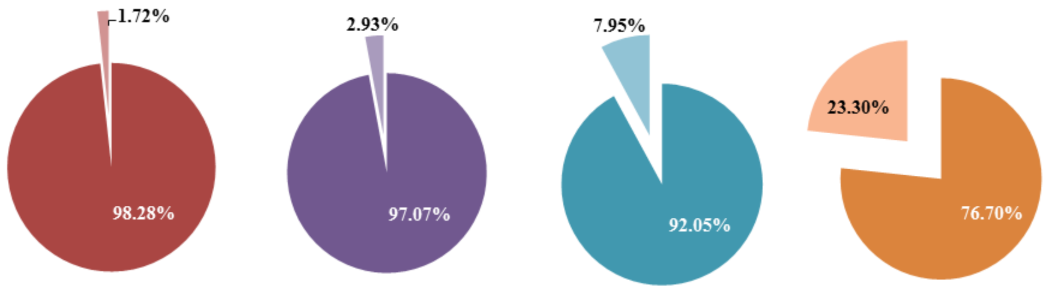
Figure 5.5: The *Pairs Completeness* for various blocking strategies. The darker color corresponds to PC, and the lighter color corresponds to the percentage of *True Matches* that were not flagged for further evaluation and therefore incorrectly classified as *Predicted Non-matches*.

of record pair comparisons is sufficiently small that it can be stored in system memory, we use a hash table for record pair generation because it is the faster method. However, when the number of record pair comparisons exceeds system memory and the database method is required , it is explicitly denoted by a "DB" marker.

All experiments in this chapter were performed on computer ϵ with the exception of the record pair generation steps that required the use of a database. These computations were performed on computer θ. Further details on the computation resources can be found in Table 2.1.

5.4 Results

We consider various parameter settings for each method, but only show the results for the best (as indicated by the *F-score* of PC and RR) parameter settings for each method in Figures 5.5 and 5.6 and Table 5.1. Figures 5.5 and 5.6 depict the PC and RR values, respectively. The *F-score* of the PC and RR values for the methods are shown in Table 5.1. Further details on various parameter settings for each method are provided in Sections 5.4.1 through 5.4.3.



(a) HLSH. (b) JLSH. (c) Canopy. (d) Token.

Figure 5.6: The *Reduction Ratio* for various blocking strategies. The darker color corresponds to the RR and the lighter value corresponds to the percentage of possible record pair comparisons that are required.

	F-score(PC,RR)
HLSH	0.9864
JLSH	0.9797
Canopy	0.8841
Token	0.9768

Table 5.1: *F-score*, as defined in Equation 5.3, of PB methods. The *F-score* reflects the parameter settings that provide for the highest PC value.

5.4.1 Additional Results for the *HLSH* and *JLSH* methods

We now examine how the parameters ϕ and μ affect the output record pairs (Figure 5.7) and running time (Figure 5.8). We consider values of $\{8, 10\}$ for ϕ in JLSH and values of $\{14, 16, 18, 20\}$ for ϕ in HLSH. We consider values of μ in $\{0, \ldots, 100\}$, as shown on the horizontal axis. We believe that these parameter settings provide a sufficient view of the trends associated with different parameters.

5.4.2 Additional Results for the *Canopy* method

Table 5.2 provides the results associated with various parameter settings tested for the *Canopy* approach. To further explore the behavior of the *Canopy* approach applied to RBFs, we select a sample of 100 records from datasets A and B such that each sample's *True Match* is included in the opposing dataset. We then calculate the Hamming distance between all record pairs. These results are shown in Figure 5.9 and provide insight on why

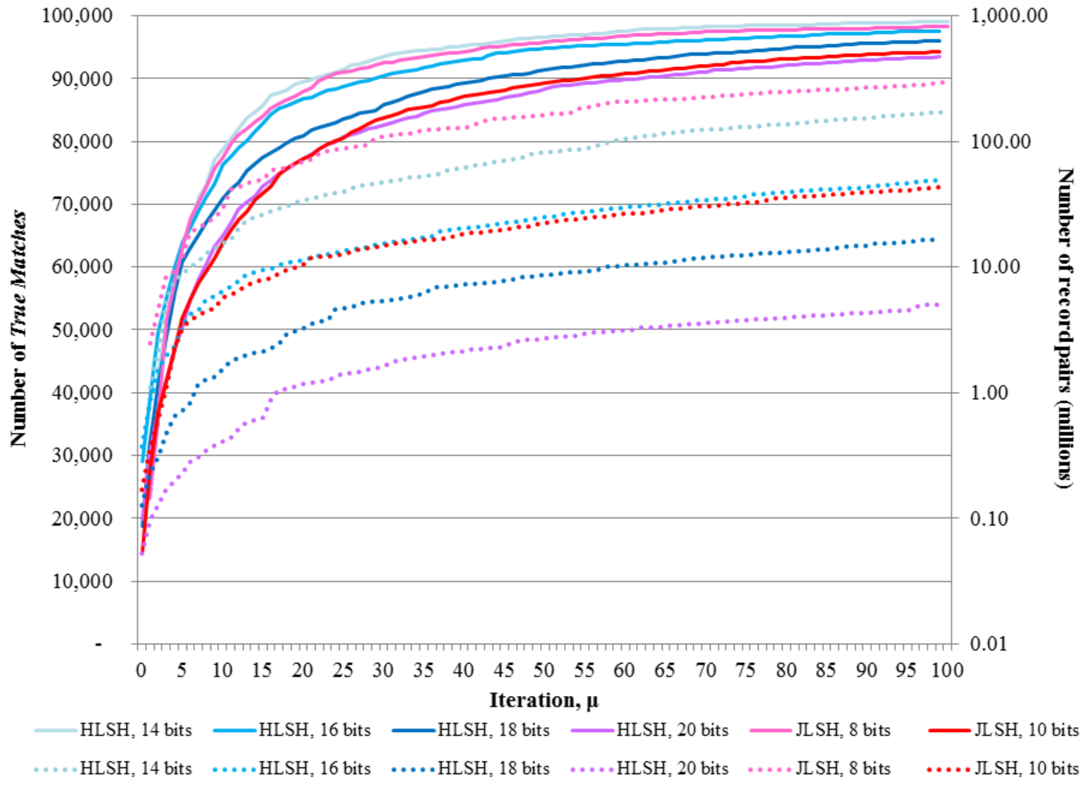


Figure 5.7: The performance of various parameter settings for the LSH methods. The number of record pairs generated for each parameter setting, shown in dotted lines, reads from the right axis on a log scale. The number of *True Matches* amongst the output record pairs, shown in solid lines, reads from the left axis. HLSH variations are shown in shades of blue, and JLSH variations are shown in shades of red.

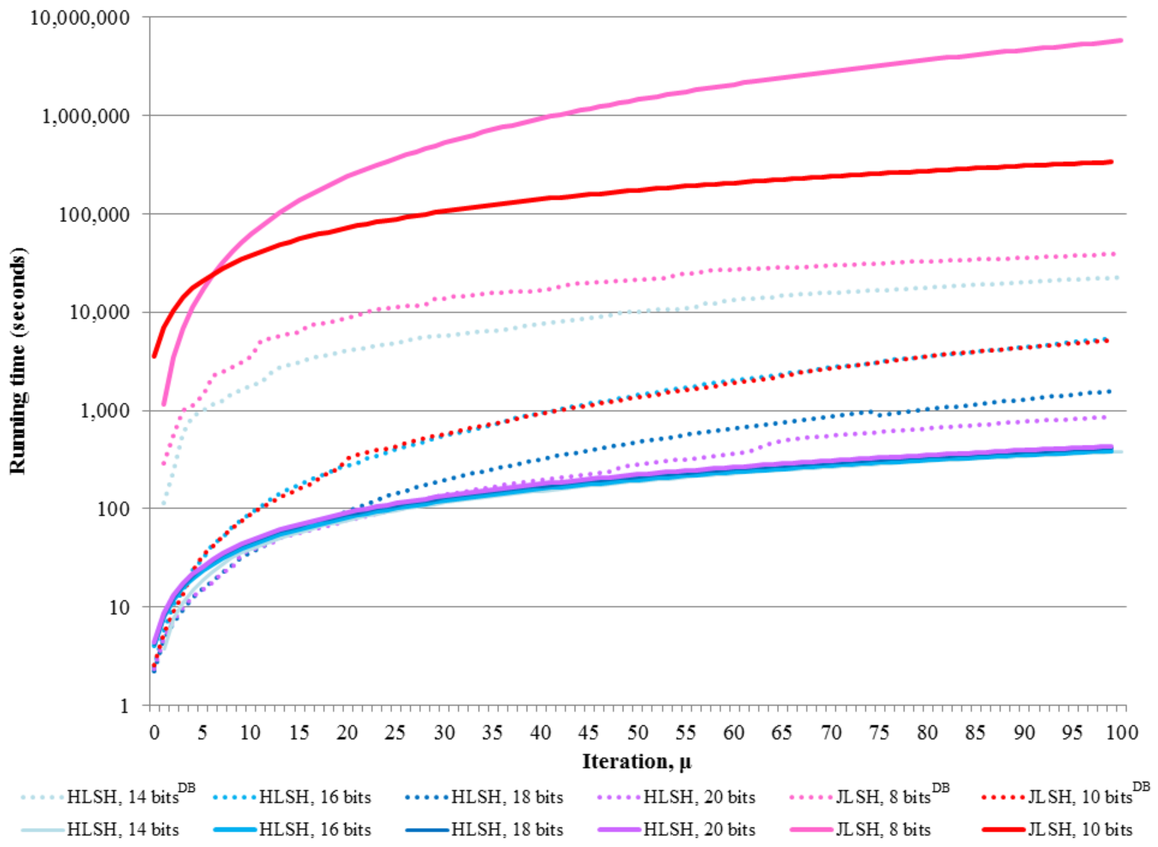


Figure 5.8: The running times of LSH methods. The time to create partitions and generate record pairs are shown with solid and dotted lines, respectively. HLSH and JLSH variations are shown in shades of blue and red, respectively. Please note the use of a log scale.

Parameters			Results		
ϕ	t_1	t_2	Number of partitions	Number of *True Matches*	Number of record pairs
10	4	2	50	2,243	104,334,264
10	3	1	275	2,477	82,713,649
10	2	0	1,024	3,938	48,204,773
20	4	2	9,595	85,043	794,646,563
20	3	1	198,131	1,881	1,332

Table 5.2: Details for various parameter settings for the *Canopy* method.

Field	Number of *True Matches*	Number of record pairs	Record pair generation time (seconds)
Surname	81,496	610,358,017	64,466
Forename	16,034	601,443,824	437,865
City	2,282	665,453,763	234,990
Street address	174	452,629,042	312,492
Sum	99,986	2,329,884,646	1,049,813

Table 5.3: Token results broken down by iteration.

the results of the *Canopy* approach, applied to RBF encodings, does not perform as well as expected based on the experience of other users who applied it successfully [20, 22, 36, 39]. Essentially, in the RBF encoding space, *True Matches* are very close (the average Hamming distance is 114 $\pm$ 50 for *True Matches* in the sample) to one another, yet far from all other records (the average Hamming distance is 470 $\pm$ 39 for *True Non-matches* in the sample). Based on the evidence, it appears that the *Canopy* approach functions best when applied in a space composed of a more continuous distribution of distances. We discuss this issue in further detail in Section 5.5.1.

5.4.3 Additional Results for the *Token* method

Recall that the *Token* method is applied in an iterative fashion such that if the first letter of field 1 agrees, the record pairs are compared; if the first letter of field 2 agrees, the record pairs are compared; and so on. Table 5.3 provides a breakdown, by iteration, of the results.

The results are presented in the order in which they were run, and only record pairs not previously identified in prior iterations are reported. In the first pass, 610,358,017 record pairs agree on first letter of *Surname*. Of these record pairs, 81,496 are *True Matches*. In

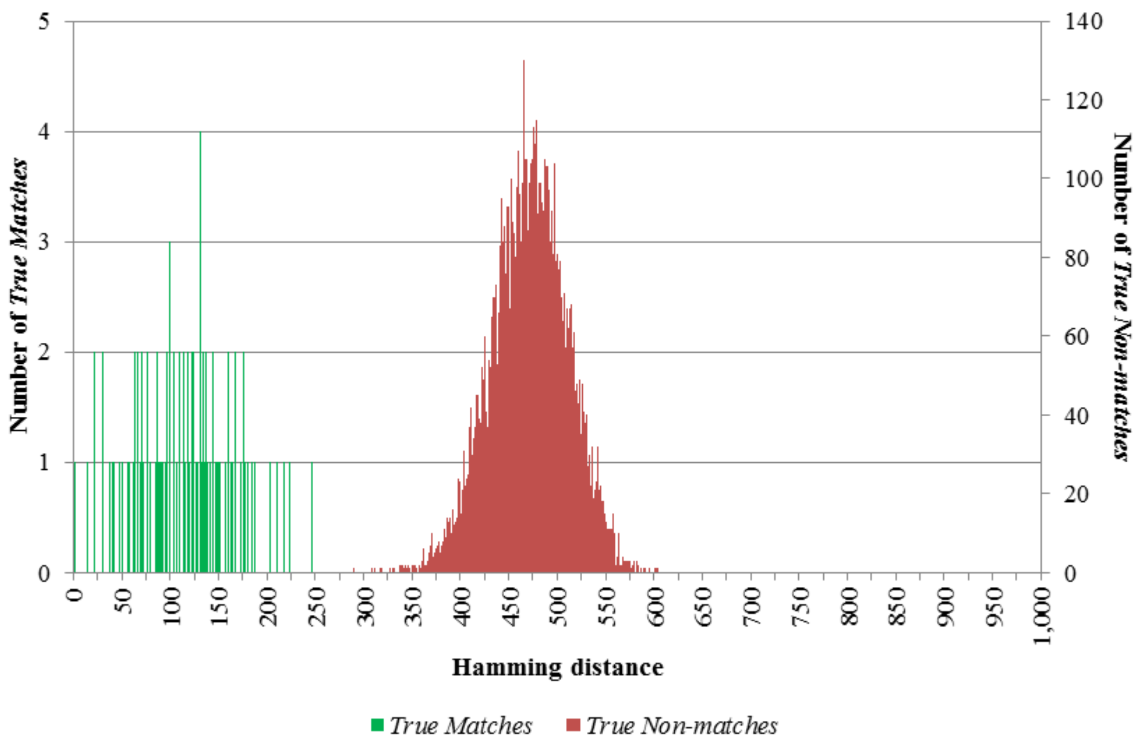


Figure 5.9: The hamming distance from a sample of records. *True Matches* read from the left axis, and *True Non-matches* read from the right axis.

the second iteration, 601,443,824 record pairs, that had not already been output, agreed on first letter of *Forename*. Of these, 16,034 *True Matches* were not identified in the previous iteration.

5.4.4 Summary of Results

The experimental analysis yields several core findings of note. First, both of the LSH-based methods considerably reduce the number of record pair comparisons required while correctly flagging the majority of *True Matches* for further evaluation. However, the time JLSH requires to partition the space is several orders of magnitude larger than HLSH. Second, the *Canopy* approach does not perform as well as expected, and we suspect this is because the distribution of distances of RBFs is not ideal input for this approach. Finally, the *Token* approach correctly flags nearly all (99.986%) of the *True Matches* for further evaluation, but also requires a significant number ($>$ 2 billion) of record pair comparisons and the runtime of this algorithm itself (> 1 million seconds, or over 291 hours) is significantly slower than other approaches.

5.5 Discussion

We first discuss the results for the control measures, followed by a discussion of the experimental measure, HLSH.

5.5.1 Controls

The control measures – JLSH, *Canopy*, and *Token* – were evaluated to provide a reference standard, and are discussed in the remainder of this section.

JLSH

The JLSH approach achieves a PC of 98.27%, meaning that it correctly flagged 98.27% of the *True Matches* for further evaluation. Additionally, it removes 97.06% of the possible record pairs from consideration, greatly reducing the number of record pair comparisons required. However, the time required to create partitions under this algorithm is non-trivial. As shown in Figure 5.8, the partition time is on the order of hundreds of thousands to millions of seconds. This is because, at each iteration, all encodings are randomly permuted ϕ times, and subsequently scanned from left to right until a value of '1' is reached. However, we implemented a more rapid implementation that does not actually permute the encodings, but accesses the bit value in the original array specified by the permutation. The running times shown reflect this quicker implementation. However, partitioning still takes orders of magnitude longer than partitioning under HLSH.

Canopy

While the *Canopy* approach has been successfully applied as a blocking technique, it is highly dependent upon its parameters (*i.e.*, the distance or similarity function, the thresholds t_1 and t_2, and the input data structure). This approach begins by centering a partition at each record and comparing this centroid to all other records. Records that are "sufficiently close" (*i.e.*, within a distance less than t_2), are grouped with the centroid represented by the initial record and are removed from the list of records which are available to serve as partition centroids. However, in the worst case, this leads to a record centered at each block. Since each block centroid is compared to all records, the *Canopy* approach has a worst case complexity of $|A| \times |B|$, which is equivalent to not performing blocking at all.

We now describe the parameter values we evaluate for the *Canopy* approach. We use Hamming distance as the distance metric, and test various values for t_1, t_2, and the input data structure. For the input data structure, we consider random samples of size $\phi = \{10, 20\}$ from the RBF. However, the results yielded either very low *PC* or low *RR*. To explore the factors that cause this problem, we analyze the distances from an arbitrary record to all other records. Figure 5.9 shows the similarity of the *True Matches* and *True Non-matches*. Essentially, the encoded data is very similar to its *True Match* and very dissimilar to all other records. While this property of the encodings is highly desirable for record linkage, it is not conducive to an efficient application of the *Canopy* approach, which performs better when the function of distances between record pairs is more smooth. Figure 5.9 suggests that the RBF landscape is sparsely populated, like the sample landscape shown in Figure 5.10.

Several factors explain why the RBF landscape is sparsely populated: 1) While records are expected to be similar on any single field value (*e.g.*, it is not uncommon for individuals to have the same *Forename*), records are generally not expected to be similar on the composition of all field values (*e.g.*, it is uncommon for individuals to have the same *Forename*, *Surname*, *City*, *Street address*, and *Gender*). This is, in fact, exactly what the RBF encoding is – a composition of all field values. Therefore, we believe that the *Canopy* approach is better suited for inputs that have a more uniform (as shown in Figure 5.3(b)), as opposed to sparse (as shown in Figure 5.10), distribution of distances amongst records. 2) Another property that contributes to the sparsity of the multidimensional space that

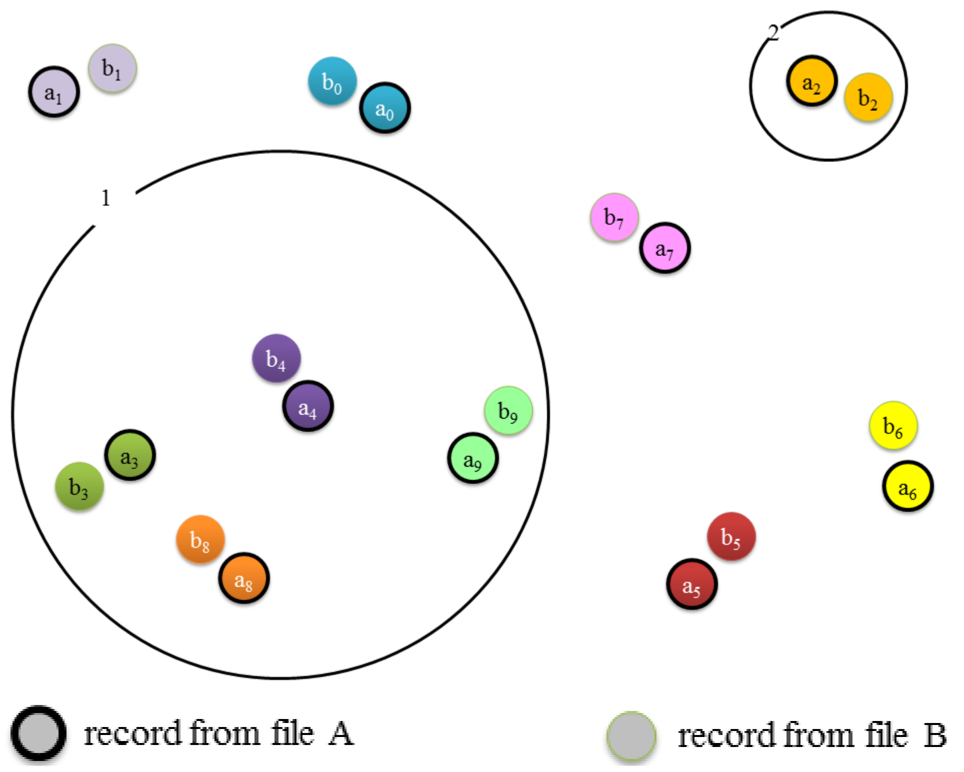


Figure 5.10: An example of a sparse distribution of records in a multidimensional space akin to that representative of the RBF encodings.

characterizes RBFs is the manner by which RBF encodings are created. Recall that when FBFs are dynamically sized, P(bit set) = 0.5. Therefore, on average, P(bit agrees) = 0.25, and P(ϕ bits agree)= 0.25^{ϕ} and, it is unlikely that RBFs will be near to one another by chance. This makes parameter selection very difficult. If a large threshold is chosen, such as is shown for Partition 1 in Figure 5.10, a significant number of record pair comparisons will be required. On the other hand, if a small threshold is chosen, such as Partition 2 in Figure 5.10, each record, or pair of records, will serve as a block centroid, which will result in $|A| \times |B|$ record pair comparisons and insignificant computational savings.

In summary, we believe the *Canopy* approach is better suited for inputs that have a more uniform distribution of distances between records, in contrast to the RBF encodings used in this work, which are sparsely distributed.

Iterative token

This approach partitions records by the first letter of the field, resulting in a hard partition of the space. However, this approach is applied iteratively, using multiple fields as the blocking variable, resulting in a fuzzy partition. While this approach yields a very high PC value of 99.986%, we have several concerns with the application of such a technique in practice. First, it outputs over 2 billion record pair comparisons on this dataset. Second, the algorithm itself is computationally intensive. While it takes just seconds to create the partitions in this example, it takes over a week to loop over the very large partitions to create the list of record pairs. There are two reasons this step takes so long. The first is that block sizes may be excessively large because block sizes depend on the frequency distribution of the blocking variable. For example, partitions based on the first letter of *Surname* will have a frequency distribution similar to the one shown in Figure 5.1. Secondly, many record pairs exist in multiple blocks. For example, if all field values are recorded correctly, the *True Match* corresponding to an individual will be placed in the same partition for each field value invoked to partition the space.

Recall that the fields *Ethnicity* and *Gender* were not applied as a blocking variable for this algorithm. The reason for this decision is that the domains of these fields is small (*i.e.*, *Gender* can take only two values), such that the partitions based on these fields do not result in a significant reduction in the number of record pair comparisons. For example, if blocking is not used, $|A| \times |B|$ record pair comparisons are required. If *Gender* is used as a blocking variable, one expects $\frac{|A|}{2} \times \frac{|B|}{2}$ record pair comparisons, resulting in a reduction ratio of only 25%. Therefore, feature selection is important for this blocking approach. Fields that are recorded with high accuracy, yet take on a large number of values, will provide for the greatest reduction in the number of record pairs required.

In summary, the *Token* blocking approach is accurate, but does not result in significant time savings because it has a large running time and outputs a large number of record pair comparisons. Moreover, and perhaps most importantly, this method is not privacy-preserving.

5.5.2 Experimental algorithm: HLSH

Now we discuss the results for the experimental algorithm that uses a class of LSH functions which approximate Hamming distance. Recall that an ideal blocking algorithm maintains accuracy, significantly reduces the number of record pair comparisons required, and runs efficiently. HLSH achieves all of these objectives. When the parameter settings $\phi = 14$, $\mu = 100$ are used, HLSH yields a PC value of 99.02% and a RR of 98.2%. The associated *F-score*, 0.9864, is the highest of all approaches evaluated. Additionally, this algorithm satisfies the properties of a private blocking algorithm. Specifically, 1) it does not require knowledge of the plaintext values to create partitions. Rather, it operates on the secure RBF encodings. 2) The properties of the blocks do not reveal information about the plaintext values. While the an attacker can observe the frequency distribution of records in blocks, he does not have access to a mapping table that can link this information back to the plaintext values.

The HLSH algorithm has two parameters: ϕ and μ. In the remainder of this section, we discuss parameter selection.

Parameter selection

The parameter ϕ corresponds to the number of bits that randomly sampled. As ϕ grows, more bits are required to agree in value, so fewer records will end up in each partition. As a result, the number of record pair comparisons will be lower. However, if ϕ becomes too large, the accuracy may decrease because even *True Matches* may not agree on all bits and will therefore be placed in different partitions.

In contrast, as μ grows, the number of record pair comparisons grows, but the accuracy may improve. This is because if a *True Match* was not equivalent on the previously selected set of ϕ bits, it may agree on the set of ϕ bits selected in the next iteration.

Stopping criteria

We now discuss an approach to determine how many iterations μ to perform, to inform the parameter selection process in practice. In short, in each iteration, the method will capture an additional similar record pairs that were not identified in previous iterations. Therefore, when the rate at which the method captures additional record pairs becomes very small, iterating can cease because most of the similar record pairs in the dataset have been identified.

5.6 Conclusions

This chapter presented a method, HLSH, to support private blocking. This method uses a class of LSH functions, that approximate Hamming distance, based on bit sampling. HLSH accepts as input the RBF encodings corresponding to a set of records and outputs the set of record pairs that should be considered for classification in PRL. HLSH performed on RBF encodings is privacy-preserving in the sense that it does not require knowledge of the plaintext values, and there is no clear way to use properties of the partitions to determine information about the plaintext values. This method runs more quickly than all other methods evaluated and also significantly reduces the computational complexity of PRL while maintaining a high degree of accuracy.

5.6.1 Future Work

In Chapter 4, we bring together the ideas presented throughout this dissertation into a single "cradle-to-grave" framework for PRL and evaluate this framework on a real medical dataset.

CHAPTER VI

A FRAMEWORK FOR PRIVATE RECORD LINKAGE

6.1 Introduction

The work described in this chapter brings together all the ideas presented in previous chapters into a single cradle to grave framework for PRL. We have called it “cradle to grave” because it begins with a set of records held by two data holders, presents a method for each step of the PRL process, and outputs a set of *Predicted Matches* and *Predicted Non-matches*. In this chapter, we also address some of the practical issues associated with PRL (and record linkage in the more general sense), such as how to handle missing data. We evaluate several experimental approaches, based on the work presented in this dissertation, and compare them to several control approaches.

In addition to bringing together the methodology presented in previous chapters, we introduce a new method for field and record pair comparison that is enabled by the RBF encodings, which we hypothesize will result in time savings. Specifically, we propose using the similarities of the RBFs, calculated via the Dice coefficient (Equation 3.2), as a comparison of the records. This eliminates the need for a distinct record pair comparison step. The proposed PRL framework is shown in Figure 6.1.

6.1.1 Chapter Outline

This chapter is organized as follows. In Section 6.2, we summarize the related work and specifically consider how others have handled missing data values in record linkage applications. Section 6.3 describes the methods used in this chapter and presents ways for handling missing data values. This section also describes the control methods evaluated, the datasets used, and the evaluation metrics utilized. Section 6.4 presents the results of the evaluation, and Section 6.5 discusses their implications. Finally, Section 6.6 concludes and summarizes

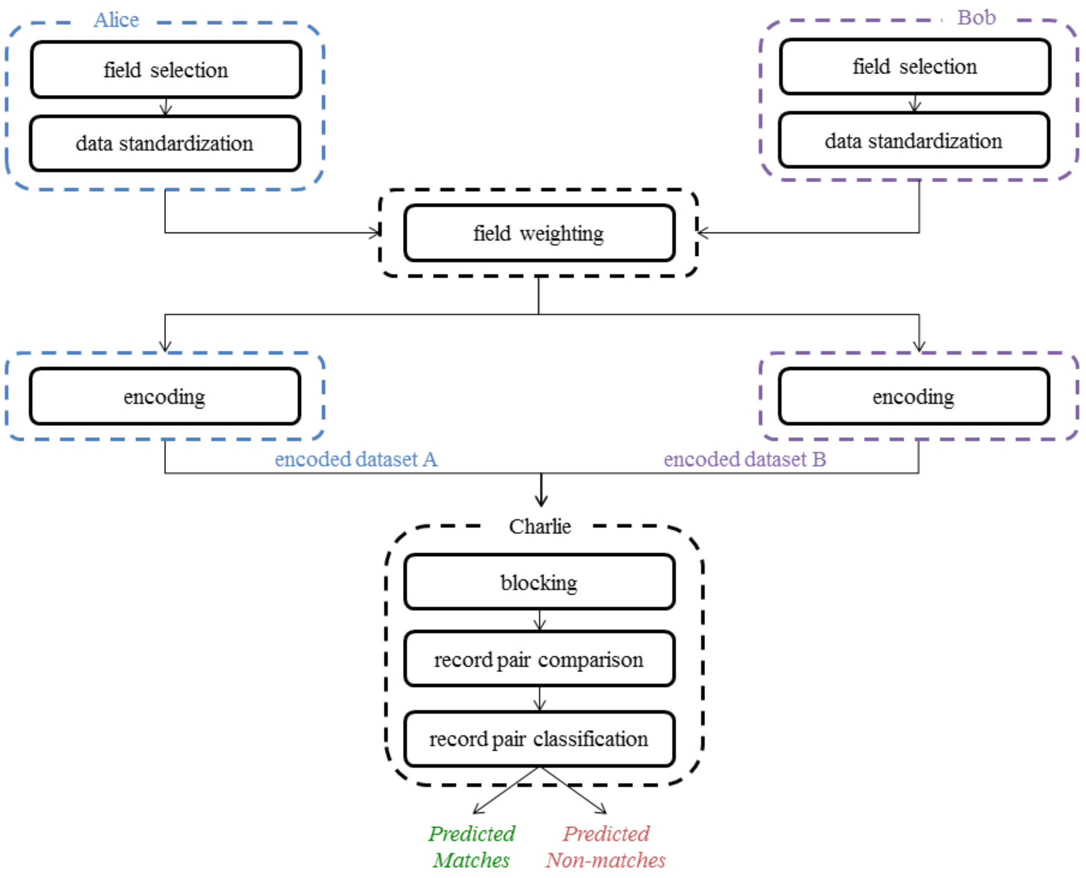


Figure 6.1: A workflow diagram for the proposed PRL framework.

the chapter.

6.2 Related Work

One major challenge associated with record linkage, in practice, is how to handle records with missing field values. In this section, we describe work done by others on handling missing field values in record linkage.

One approach to handling missing data values is to eliminate them by returning to the source to collect the information, as described by Winkler for a linkage of U.S. Census data [120]:

> *"In a very large 1990 Decennial Census application, the computerized procedures were able to reduce the need for clerks and field follow-up from an estimated 3000 individuals over 3 months to 200 individuals over 6 weeks. The reason for the need for 200 clerks is that both first name and age were missing from a small proportion of Census forms and Post Enumeration Survey forms. If the clerks cannot supply the missing data from auxiliary information, then a field visit is often needed to get the missing information."*

However, this requires significant resources and is not always feasible. For example, in the application in question, hundreds of individuals were required to collect the missing data values. Additionally, in the context of medical data, contacting individuals to fill in missing values may not be appropriate.

Another approach for handling missing field values, in the context of record pair comparison using the FS field weights (see Figure 2.6), is to assign missing fields a weight of 0, rather than the agreement or disagreement weight [48]. This practice for handling missing field values was used in the evaluation described in Chapter 3.

Regardless of how missing field values are handled, it is well-documented that they are a considerable challenge for record linkage. Moreover, records where all data fields are not present are more likely to be misclassified [33, 89, 105].

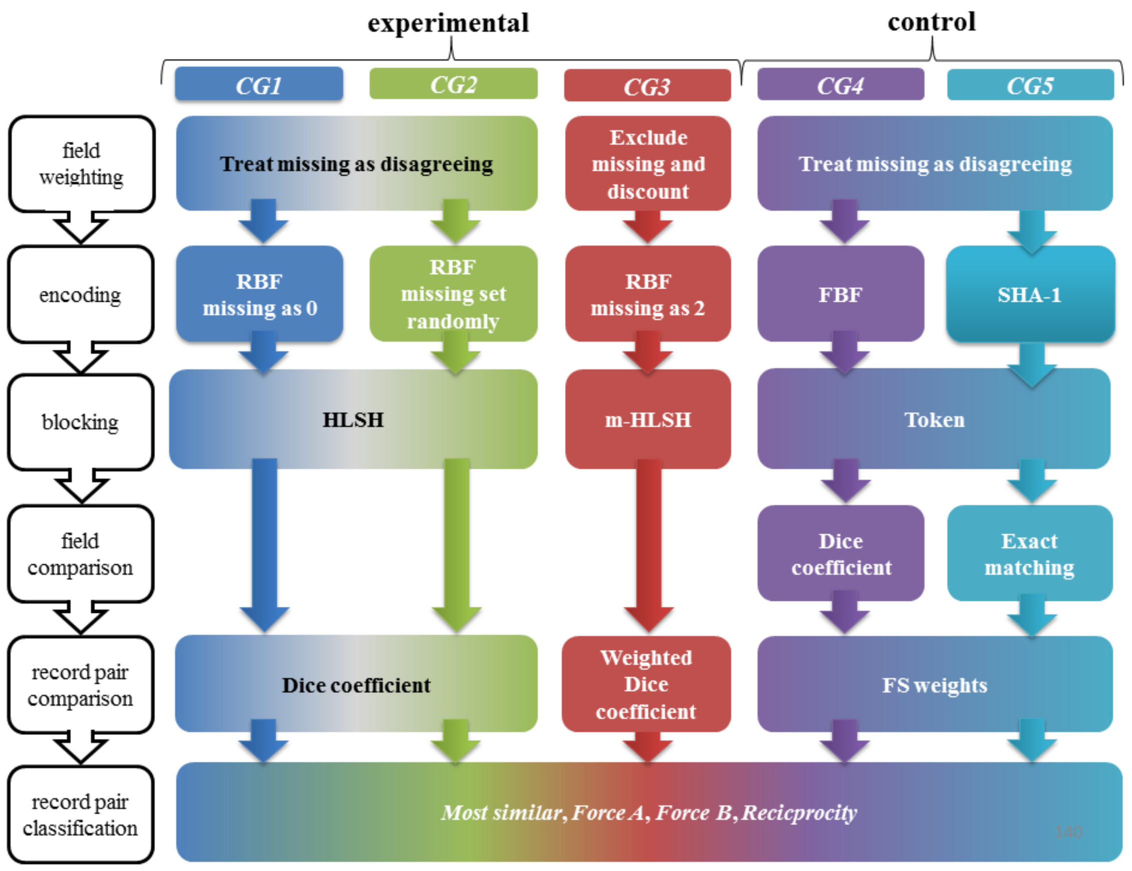


Figure 6.2: A component-based summary of the specific methods and strategies applied in each variation of the proposed private record linkage framework.

6.3 Materials & Methods

In Section 6.3.1, we discuss three different methods for encoding missing field values in RBF encodings. These will be compared to two control methods, described in Section 6.3.2. All methods, and how different steps of the record linkage process relate to each, are summarized in Figure 6.2.

6.3.1 Experimental Methods

In this section, we describe the three experimental methods, *CG1* through *CG3*, which are distinctive because they are based on three different methods for encoding missing field values in RBF encodings.

Treatment of missing data

Missing data transpires when a field value is not documented by the data holder. This may happen because the information is unavailable or when an individual lacks certain features (*e.g.*, not all people have a *Middle name*). In this section, we discuss how missing data influences different aspects of the record linkage process.

Missing data and field weighting

The FS field weighting algorithm, described in Section 2.1.6, uses the conditional probability of field agreement given match status to assign a weight to each field. Specifically, two values are calculated for each field: $m[i]$, the probability the i^{th} field value agrees amongst record pairs that are *True Matches*; and $u[i]$, the probability the i^{th} field value agrees amongst record pairs that are *True Non-matches.* However, this algorithm does not specify how missing field values should factor into the algorithm. As alluded to in Section 6.2, the literature on this topic is sparse.

We consider two options, which we call *Treat missing as disagreement* and *Exclude missing and discount.*

Treat missing as disagreement

The FS field weighting algorithm takes as input the set of γ vectors, which represent the field similarities of record pairs. The first option is to treat a missing field value as a disagreement when comparing fields to create the γ vectors. Essentially, any field value, when compared to a record where that field value is missing, receives a similarity of 0. In this method, the discriminatory power of fields is implicitly discounted because it is not seen to agree as often. This is in contrast to the *Exclude missing and discount* method, described below, where field weights are explictly discounted. The *Treat missing as disagreement* approach is used in field weighting for experimental methods *CG1* and *CG2*, as shown in Figure 6.2.

Exclude missing and discount

Another option is to only input records in which all field values are present to the FS algorithm. However, this may lead to an overestimate of field discriminatory power. If the

	Surname	*Forename*	*City*	*Street*	*Gender*	Sum
range	10	8	3	12	2	
% values missing	25%	50%	0%	67%	50%	
$range_{discounted}$	7.5	4	3	4	1	
w	38%	21%	15%	21%	5%	100%

Table 6.1: Handling missing data in field weighting: Exclude missing and discount.

weighting method estimates a field's discriminatory power using only samples where the field value is present, but the field value is missing most of the time in practice, then the estimates of the field's discriminatory power will not be accurate. Therefore, we propose discounting the weights based on only complete records by the percentage of missing values for each field. An example is shown in Table 6.1. This approach is adopted by experimental method *CG3*, as shown in Figure 6.2.

Encoding missing data in Bloom filters

The previous section considered how to determine field weights in the face of missing data. Now we consider various ways missing data can be encoded in Bloom filters. To demonstrate various techniques, throughout this section, we use two sample Bloom filters, α and β. These encodings are RBFs and contain bits drawn from three different fields, each shown in a different color.

$$\alpha : |0|1|0||0|1|1|0|1||1|0|1|1|$$

$$\beta : |1|1|0||1|0|1|0|0||1|1|0|1|$$

Now, we consider the case when the corresponding field value for the bits shown in red are missing in the record encoded in β. We can choose to encode the missing values in three ways: 1) let the bits remain at their initial value of '0', 2) randomly set the bits corresponding to missing field values, or 3) explicitly denote the bits using a non-binary value.

Case 1: Missing bits are encoded as '0'

If the bits corresponding to the missing field are encoded as '0', β will be presented as follows.

$$\beta_{missing,1} : |1|1|0||0|0|0|0|0||1|1|0|1|$$

This is the strategy adopted by experimental framework *CG1*.

Case 2: Fill in missing bits at the expected frequency

In Section 4.3.1, we introduced a method to dynamically size FBFs such that $p\%$ of bits are set on average. We selected $p = 0.5$ to maximize entropy. Based on this strategy, we propose to randomly set 50% of the bits corresponding to missing field values to a value of '1'. When this approach is applied, β is represented as follows:

$$\beta_{missing,2} : |1|1|0||1|0|1|0|0||1|1|0|1|$$

Thus, when comparing two encodings, P(bit agrees) = P(bit set) $\times$ P(bit set) = 0.5 $\times$ 0.5 = 0.25 and, we expect a background agreement rate of 25%. By randomly setting $p\%$ of bits corresponding to missing field values, we essentially bring the expected agreement rate for these bits up to the background agreement rate. In contrast, leaving these bits unset means that the background agreement rate associated with missing field values is 0%. Therefore, this approach brings missing field values in line with the background agreement rate generally seen when comparing RBF encodings.

Case 3: Missing bits are explicitly denoted

We can also represent the missing field values using some non-binary character, such as a wildcard '*'. In this case, β is represented as follows:

$$\beta_{missing,3} : |1|1|0||*|*|*|*|*||1|1|0|1|$$

This is the strategy adopted by experimental framework *CG3*.

Later in this section, we will consider how these various encoding forms can be compared to one another.

Missing data in blocking

Chapter 5 presents a method, HLSH, for private blocking using Hamming distance based on LSH families. We evaluate this technique in the context of a full PRL application. We

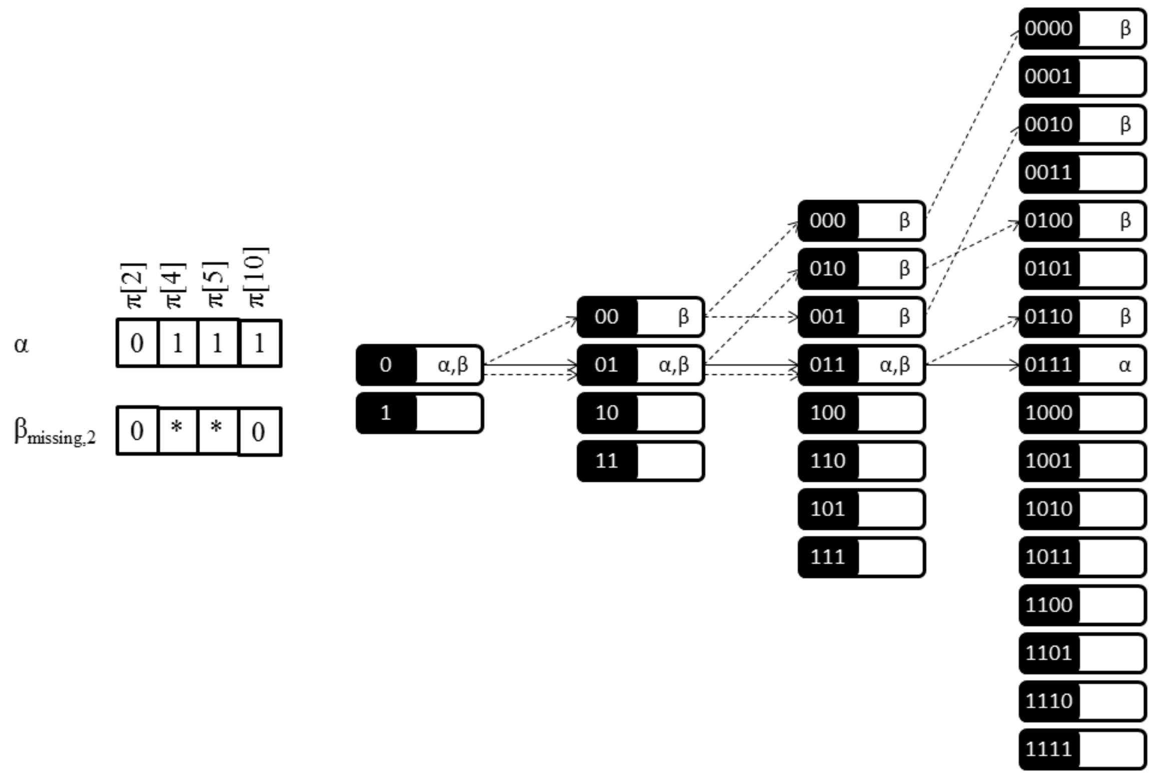


Figure 6.3: An example application of the m-HLSH algorithm. Bit indices 2, 4, 5, and 10 are selected during the first four iterations. $\pi[\rho]$ indicates the ρ^{th} bit in RBF π. The trajectory of α is shown with a solid line, and the trajectory of $\beta_{missing,2}$ is shown with a dotted line.

also introduce a variant of this approach, m-HLSH, that can be applied when missing data is explicitly denoted, using a non-binary character, in Bloom filter encodings.

The m-HLSH variation works as follows. Recall that ϕ hash functions are applied at each iteration μ. Each hash function essentially selects a random bit index. Typically, the record is placed into a partition corresponding to the value of the bit at this index, which is either '0' or '1'. The principle behind m-HLSH is that when this bit value is missing, the record is placed into the partitions corresponding to both '0' and '1'. For illustration, suppose that encodings α and $\beta_{missing,2}$ are input to m-HLSH and that bit indices 2, 4, 5, and 10 are selected during the first four iterations. The corresponding bit selections, a step-by-step picture of how the algorithm works, and the output are shown in Figure 6.3.

As can be seen, when π_m bits are missing in encoding π, the record will be placed in 2^{π_m} partitions at each iteration. We propose using multiple iterations to provide additional opportunities to identify *True Matches* that were not identified in previous iterations. If μ is the number of iterations, each record is placed in $\mu \times 2^{\pi_m}$ partitions in total. As an example, when 0 bits correspond to missing values, each record is placed in a single partition at each iteration, or μ partitions in total. When the number of missing values is high, the number of record pairs resulting from the blocking algorithm increases exponentially.

When missing data is explicitly denoted using a non-binary value (*i.e.*, experimental method *CG3*), m-HLSH is used for blocking.

<u>Missing data and encoding comparison</u>

The way in which missing data is encoded in the Bloom filter affects how the resultant Bloom filters can be compared. For illustration, we refer back to the sample Bloom filters α, $\beta_{missing,1}$, $\beta_{missing,2}$, and $\beta_{missing,3}$. According to Equation 3.2, the actual similarity of record pair (α,β) is $2 \times \frac{4}{7+7} = 0.57$.

Case 1: Missing bits are encoded as '0'

When the Dice coefficient is applied, the similarity of the record pair (α, $\beta_{missing,1}$) is $2 \times \frac{3}{7+5} = 0.5$. This reflects that the record pair is less similar than it actually is. The Dice coefficient is used to measure encoding similarity in experimental method *CG1*.

Case 2: Fill in missing bits at the expected frequency

The standard Dice coefficient can be applied in this case as well, resulting in a similarity of $2 \times \frac{4}{7+6} = 0.62$ for record pair (α, $\beta_{missing,2}$), a measure of similarity that is actually higher than the true value in this case. The Dice coefficient is used to measure encoding similarity in experimental method *CG2*.

Case 3: Missing bits are explicitly denoted

When bits corresponding to missing field values are explicitly denoted, we introduce a weighted form of the Dice coefficient, shown in Equation 6.1, that can be used. Let $|\alpha|$ denote the number of bits in the Bloom filter encodings, let ρ_m indicate the bits corresponding to a missing field value in Bloom filter ρ. Then, the weighted Dice coefficient, wd, is defined as:

$$wd(\alpha, \beta_{missing,3}) = \frac{d(\alpha, \beta) \times (|\alpha| - |\alpha_m \cup \beta_{missing,3,m}|)}{|\alpha|} \tag{6.1}$$

Essentially, the Dice coefficient is weighted by the number of bits representing a non-missing field value. In this example, the weighted Dice coefficient of record pair (α,$\beta_{missing,3}$) is $\frac{0.5\times7}{12}$ = 0.29. The weighted Dice coefficient is used to measure encoding similarity in experimental method *CG3*.

Methods for each step

In this section, we describe the proposed experimental methods used at each step of PRL. The methods are summarized in Figure 6.2. We call the experimental methods for the cradle to grave evaluation $CG1$, $CG2$, and $CG3$.

<u>Field selection</u>

When selecting fields, it is essential to select a sufficiently large number of fields to uniquely identify an individual. Selecting fields of high quality, that are consistently and accurately recorded, is also important. For the experiments performed in this chapter, we utilize a dataset, described further in Section 6.3.4, corresponding to patients at Vanderbilt University. While additional fields, such as SSN, were available in this dataset, we believe it is important to create a dataset representative of those typically encountered in PRL applications. Unique identifiers[*], such as SSN, may not generally be available, so we select a subset of the fields that we believe is more representative of fields likely to be encountered in practice.

We now describe how we prepared the dataset for the cradle to grave evaluation described in this chapter. We began with six possible fields: *Surname*, *Forename*, *City*, *Street address*, *Ethnicity*, and *Gender*. However, we observed that over 50% of the records did not have an *Ethnicity* value recorded. Therefore, we did not expect this field to be useful

[*]See discussion in Section 1.2.2.

in record linkage for this dataset and so we exclude it from our empirical investigations. An alternative approach we could have utilized is to perform linkage using all fields excluding *Ethnicity*. Records not linked in this first linkage process could be input to a second linkage process where *Ethnicity* is used as a field value. This iterative approach to handling missing field values is discussed further in Section 7.2.1.

We then confirmed that all records were unique when defined over the five fields – *Surname*, *Forename*, *City*, *Street address*, and *Gender* – to ensure that a sufficient number of fields had been selected.

At this point, we made the decision to exclude records where multiple field values are missing. If a single field is missing, four fields are still available, and this information may be sufficient to classify the record. However, if more than one field is missing, a maximum of three fields are available which may not be sufficient for unique identification. If these records were carried forward in the linkage, then the parameterization of field weights may be inaccurate. In turn, this will lower the chance that records which contain sufficient information for classification will be correctly classified.

The resultant dataset is described further in Section 6.3.4.

Data standardization

In this step, we attempt to ensure that the data is represented consistently across datasets. We also identify and standardize values used to missing data, such as "NULL" and "N/A", to a single representation.

Field weighting

We calculate field weights in both ways described in Section 6.3.1 (*i.e.*, *Treat missing as disagreement* and *Exclude missing and discount*). In both cases, we use a subset of the records as input to the FS field weighting algorithm. Specifically, we use subsets of size 1,000, where each record in dataset A has a *True Match* in dataset B. This results in 1,000,000 record pairs, 1,000 of which are *True Matches* and 990,000 of which are *True Non-matches*.

	Parameters	*Surname*	*Forename*	*City*	*Street*	*Gender*
Treat missing as disagreement	m	81%	80%	79%	26%	98%
	u	0.10%	0.20%	14.00%	4.8 x 10^{-3}%	50.12%
	w_a	6.73	5.97	1.71	8.55	0.67
	w_d	-1.65	-1.58	-1.34	-0.29	-2.99
	range	8.38	7.56	3.05	8.83	3.66
	w	26.63%	24.00%	9.69%	28.06%	11.62%
Exclude missing and discount	m	81%	80%	79%	26%	98%
	u	0.09%	0.22%	13.71%	4.8 x 10^{-3}%	50.12%
	w_a	6.77	5.90	1.75	8.59	0.67
	w_d	-1.65	-1.61	-1.41	-0.30	-2.99
	range	8.43	7.51	3.16	8.89	3.66
	% values missing	0.00%	0.21%	0.15%	2.13%	0.03%
	$range_{discounted}$	8.43	7.49	3.15	8.70	3.66
	$w_{discounted}$	26.81%	23.83%	10.03%	8.69%	11.64%

Table 6.2: Field weights for the various ways of handling missing data values: *Treat missing as disagreement* and *Exclude missing and discount.*

The EM algorithm, which can be applied to estimate the m and u values in the absence of knowledge of the true match status, tends to yield inaccurate estimates when a significant number of field values are missing [14]. Therefore, in this chapter, we use our knowledge of the true match status to calculate the actual m and u values. We appreciate that obtaining accurate field weights in the face of missing data is an open research question, and we discuss this further in Chapter 7.

The resulting field weights for the *Treat missing as disagreement* and *Exclude missing and discount* methods are shown in Table 6.2. Since the number of record pairs with missing values is small, as shown in Table 6.1, there is not a significant difference between the outputs of the methods.

Encoding

In *CG1*, *CG2*, and *CG3*, we use RBFs created by weighted bit sampling from dynamically-sized FBFs, as described in Section 4.3.1 and shown in Figure 4.4. The parameters specific to this dataset are shown in Table 6.3. The length of the RBFs is 1,000 bits, and we enforce a security parameter $q = 4$. Further details on this methodology can be found in Section 4.4.3.

	Surname	*Forename*	*City*	*Street*	*Gender*
Average field length	6.4	5.9	9.2	15.8	1.0
Average # n-grams (g)	7.4	6.9	10.2	16.8	2.0
m_{RBF}	161	151	222	365	44

Table 6.3: Summary statistics for *dynamic* Bloom filter sizing where k=15, n=2, p=0.5.

		Surname	*Forename*	*City*	*Street*	*Gender*	Sum
CG1, *CG2*	$range$	8.38	7.56	3.05	8.83	3.66	
	w	26.63%	24.00%	9.69%	28.06%	11.62%	100%
	RBF composition	266	240	97	281	116	1,000
CG3	$range_{discounted}$	8.43	7.49	3.15	8.70	3.66	
	$w_{discounted}$	26.81%	23.83%	10.03%	27.69%	11.64%	100%
	RBF composition	268	238	100	277	117	1,000

Table 6.4: RBF composition for *CG1*, *CG2*, and *CG3* encodings.

In this context, we explore the three variations for encoding missing data in Bloom filters described in Section 6.3.1. Bits corresponding to missing values are encoded as '0's in $CG1$, are set with $p = 0.5$ in $CG2$, and are explicitly denoted using the non-binary value '*' in $CG3$.

The RBF composition resulting from field weights calculated under the *Treat missing as disagreement* method are used in experimental methods *CG1* and *CG2*. The RBF composition resulting from field weights calculated under the *Exclude missing and discount* method are used in experimental method *CG3*. The resulting RBF compositions are shown in Table 6.4.

Blocking

When bits corresponding to missing fields are explicitly indicated, we use m-HLSH for blocking ($CG3$); otherwise, HLSH is used.

Field comparison

Recall that when RBFs are used, no explicit field comparison step is necessary because the field values are encoded in a single unit.

Record pair comparison

The Dice coefficient (Equation 3.2) is applied to the RBFs to measure record pair similarity in experimental methods *CG1* and *CG2*. The weighted Dice coefficient (Equation 6.1) is applied in *CG3*.

Record pair classification

Recall that record pair classification corresponds to the process of determining which record pairs are *True Matches* and which are *True Non-matches*, given a measure of the similarity of each record pair. We explore four different methods for record pair classification: *Most similar*, *Force A*, *Force B*, and *Force Both*. All of these methods are heuristics we have developed because this is an area that is not well-addressed in the literature.

Most similar

In this classification method, the λ record pairs with the highest similarity values are classified as *Predicted Matches*, where λ is the expected number of *True Matches*.

Force A

Under this classification strategy, we assume that dataset A is a subset of dataset B. We create a set of potential matches by forcing each record in dataset A to link to the record in dataset B to which it is most similar. However, we know that dataset A is not actually a subset of dataset B, so we apply a threshold of 0.8. This is a tunable parameter that a user can set based on his tolerance for minimum similarity of *Predicted Matches*. Potential matches having a similarity greater than or equal to this threshold are classified as *Predicted Matches*, and *Predicted Non-matches* otherwise.

Force B

This is similar to *Force A*, but in this case, it is assumed that dataset B is a subset of dataset A. Again, a threshold of 0.8 is applied.

Reciprocity

Recall that with *Force A*, the set of potential matches is created by forcing each record in dataset A to match to the record in dataset B to which it is most similar. We call this set $Potential(A)$. A similar set, $Potential(B)$ results from the *Force B* classification method. We let the set of *Predicted Matches* for the classification method *Force Both* be the intersection of the sets $Potential(A)$ and $Potential(B)$ (*i.e.*, *Predicted Matches(Reciprocity)* $= Potential(A) \cap Potential(B)$).

6.3.2 Controls

We now describe two control methods, *CG4* and *CG5*, used to create a reference standard. The remainder of this section presents the details of each step of record linkage as relevant to the control methods.

Field selection

The same five fields used in the experimental methods are also used for the control methods (*i.e.*, *Surname*, *Forename*, *City*, *Street address*, and *Gender*).

Data standardization

The same methods use for for data cleaning and standardization in the experimental methods are applied to the dataset input to control methods.

Field weighting

The field weights calculated using the *Exclude missing and discount* method, shown in Table 6.1, are applied in control methods *CG4* and *CG5*. This is because the method described in Section 3.3.5 for handling missing field values during record pair comparison is used for the control methods. Essentially, missing field values receive neither the agreement nor disagreement weight in the similarity score. Since they are not treated as disagreeing, using the field weights calculated by treating missing values as disagreement is not logical.

Encoding

For control method *CG4*, FBFs, as described in Section 3.2.2, are used for encoding field values. The bits corresponding to missing field values are left at their initial value of '0'. The parameters used are $n = 2$, $k = 15$, and $m = 500$. In control method *CG5*, SHA-1, in conjunction with salting, is used to encode field values.

Blocking

A token-based blocking approach is included in both control methods *CG4* and *CG5*. The blocking variable corresponds to the first character of each field value, and this approach is applied iteratively over all fields, as described in Section 5.3.2.

Field comparison

The Dice coefficient, defined in Equation 3.2, is used for field comparison in control method *CG4*. Binary field comparison is used for field comparison in control method *CG5*, as described in Section 3.1.

Record pair comparison

The agreement and disagreement weights associated with the FS algorithm are applied to determine the record pair similarity score, as described in Sections 2.1.6 and 2.1.7. When a field value is missing, neither the agreement nor disagreement weight are added to the record pair similarity score. This practice is also applied in Chapter 3 and used by Gomatam *et al.* [48].

Record pair classification

The four methods for record pair classification invoked for the experimental methods – *Most similar*, *Force A*, *Force B*, and *Force Both* – are also used for the control methods.

6.3.3 Evaluation Metrics

Since the cradle to grave evaluation considers all steps of the record linkage process, we utilize different evaluation metrics at each step. The evaluation metrics are described in the remainder of this section.

Encoding

To examine how the RBF encodings are affected by the application of the security constraint $q = 4$, we calculate the number of bits available under the constraint and which fields contain bits that satisfy the constraint. We also look at the similarity of the *True Matches*, as well as a sample of *True Non-matches*, to determine which encoding methods best separate the two. Finally, we take a closer look at the set of *True Matches* containing a missing value by examining their similarity.

Blocking

We use the metrics introduced in Chapter 5, RR (Equation 5.1) and PC (Equation 5.2), to evaluate the blocking algorithms. We also examine the running times of the blocking algorithms themselves.

Record pair classification

To examine the final output of the record pair classification step, we use the *Precision* and *Recall* measures, as defined in Equations 6.2 and 6.3.

$$Precision = \frac{TP}{TP + FP} \tag{6.2}$$

$$Recall = \frac{TP}{TP + FN} \tag{6.3}$$

where TP is the number of true positives (*i.e.*, *Predicted Matches* that are *True Matches*), FP is the number of false positives (*i.e.*, *Predicted Matches* that are *True Non-matches*), and FN is the number of false negatives (*i.e.*, *Predicted Non-matches* that are *True Non-matches*).

6.3.4 Dataset

The dataset used in this chapter is composed of identifiers and demographics from real patient records. Background information on the dataset can be found in Chapter 2.4.2.

As described in Section 6.3.1, the fields *Surname*, *Forename*, *City*, *Street address*, and *Gender* are included for the record linkage study described in this chapter. Records missing more than one field value were excluded based on the earlier justification.

The properties of the resultant dataset are $|A|$ = 28,415 records, $|B|$ = 25,668 records, and $|A \cap B|$ = 19,482 records (*i.e.*, there are 19,482 *True Matches*). Additional information on the extent to which field values are absent is provided in Table 6.5. Of the

	Surname	*Forename*	*City*	*Street*	*Gender*
Number missing from A	8	10	25	244	5
Number missing from B	26	105	56	906	10
Number missing total	34	115	81	1,152	20
% missing total	0.1%	0.2%	0.1%	2.1%	0.0%

Table 6.5: Summary statistics for the field values missing in the Vanderbilt dataset.

19,482 *True Matches*, 3,051 agree on all fields. This means that approximate field comparison techniques, such as that provided by RBF encodings, are expected to be important in correctly classifying the other *True Matches*. However, it is possible that some *True Matches* are very dissimilar. Imagine, for instance, that a patient has changed her *Surname* and moved between the times her information was recorded in datasets A and B. In this example, two of the five field values will not agree. Therefore, we use control measures, described in Section 6.3.2 below, to indicate the extent to which records in the dataset are capable of being correctly classified.

6.3.5 Implementation Details

Computer ζ is utilized for encoding and creating partitions, computer θ for record pair generation, and computer ι for the remainder of the experiments described in this chapter. Additional details on these computational resources can be found in Table 2.1. While the use of multiple computers affects the specific running times reported, it does not influence the findings and conclusions of this study.

A global hash was used in the record pair generation step of blocking for HLSH because the number of record pairs generated could be stored in system memory. This was not the case for the *Token* blocking method, so a database was used in record pair generation. Further details on this can be found in Section 5.3.5.

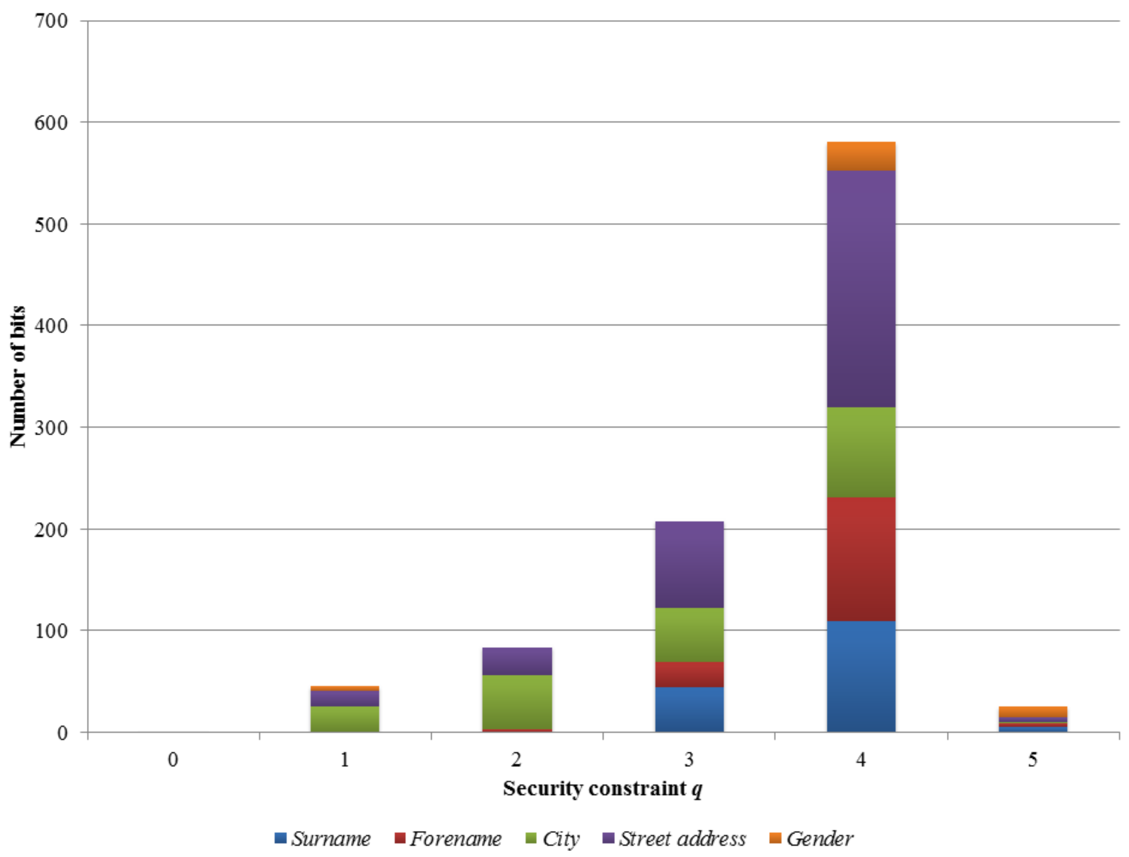


Figure 6.4: The number of bits satisfying security constraint q, where bits corresponding to missing fields are encoded as '0'.

6.4 Results

We now present the results for each step of the blocking process. Specifically we consider results with respect to the different methods of encoding and blocking missing data. Then we evaluate the final results produced in the record pair classification step.

6.4.1 Encoding Results

Figures 6.4 and 6.5 show the number of bits available, broken down by field, under each value of the security constraint q, when missing values are encoded as '0' and randomly, respectively. Under the security constraint $q = 4$, 606 and 598 bits satisfy this constraint when missing values are encoded as '0' and randomly, respectively.

To explore how the various methods for encoding missing values affect the separation

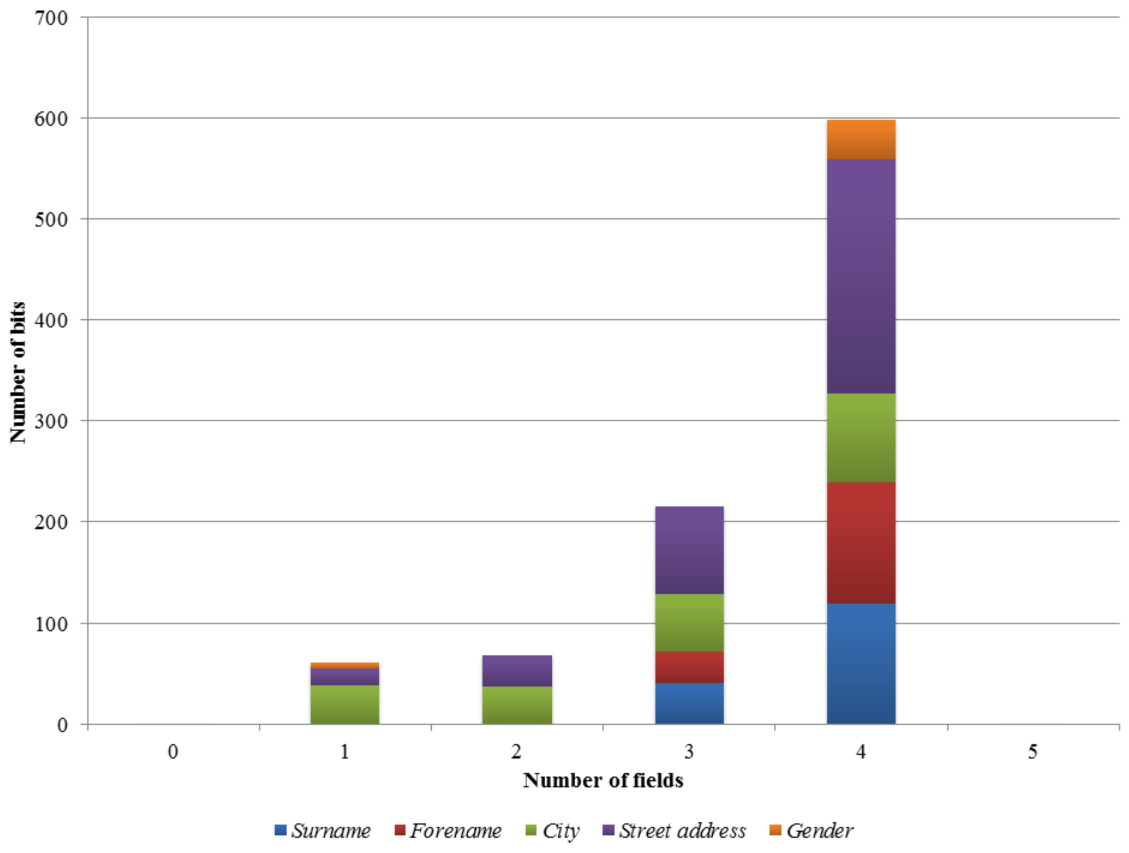


Figure 6.5: The number of bits satisfying security constraint q, where bits corresponding to missing fields are randomly set with 50% probability.

of *True Matches* and *True Non-matches*, we evaluate the similarity of the *True Matches* under each encoding method. Evaluating all *True Non-matches* is too computationally intense (there are 729,336,738 *True Non-matches*), so we evaluate a sample of 1,000,000 *True Non-matches* to estimate the similarity of the set of *True Non-matches*.

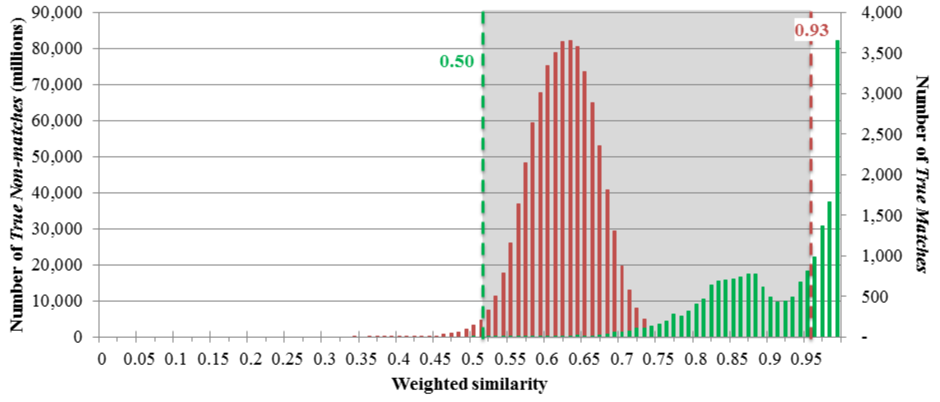
The ideal encoding mechanism separates *True Matches* and *True Non-matches* such that the *True Matches* have high similarity values and the *True Non-matches* have low similarity values. We call the similarity range that contains both *True Matches* and *True Non-matches* the "gray zone". Separating *True Matches* and *True Non-matches* is equivalent to minimizing this gray zone area. Figure 6.6 shows the similarities of *True Matches* and *True Non-matches*. The least similar *True Match* and most similar *True Non-match* are labelled and indicated with vertical green and red dotted lines, respectively. These demarcate the gray zone boundaries. Figure 6.6 indicates that the encoding mechanism used in *CG2* (size of gray zone = 0.38) better separates *True Matches* and *True Non-matches* than *CG1* and *CG3* (size of gray zone = 0.43, 0.48, respectively).

Different ways of encoding missing values

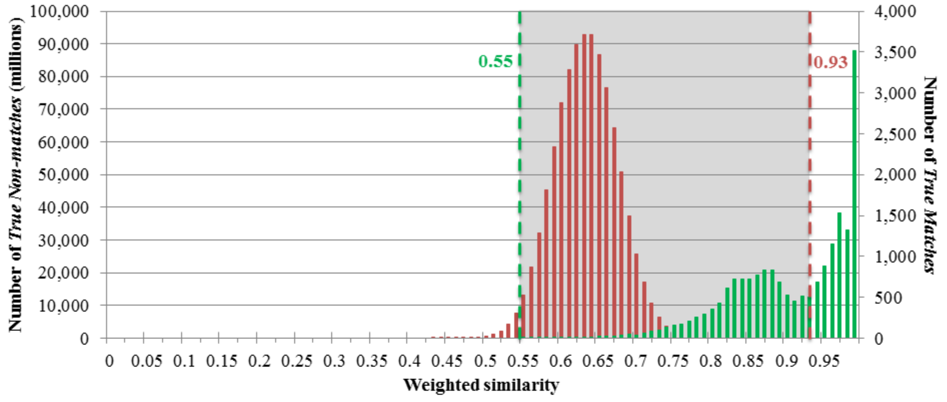
Finally, we take a closer look at the set of *True Matches* that have a missing field value by examining their similarity under the different methods for encoding missing field values. Ideally, *True Matches* would be similar to one another. Figure 6.7 illustrates that *True Matches* are most similar under the encoding methods used in *CG1* and *CG2* (the average similarities are 0.80 ± 0.07 and 0.83 ± 0.06, respectively) and least similar given the *CG3* (the average similarity is 0.68 ± 0.06) encoding method. In summary, *CG1* and *CG2* are statistically superior to *CG3*, but not one another.

6.4.2 Blocking Results

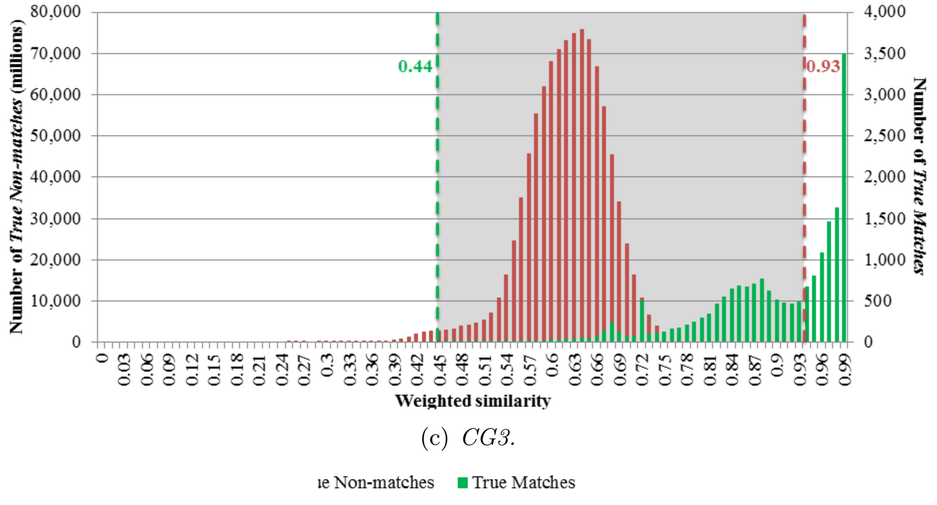
We now examine the results of the blocking step. With respect to the LSH methods, we test various values for ϕ, the number of bits in each sample and μ, the number of iterations applied. The number of record pairs output from each approach, and the number of *True*



(a) *CG1.*



(b) *CG2.*



(c) *CG3.*

Figure 6.6: Separation of *True Matches* and *True Non-matches* under different methods of encoding missing data.

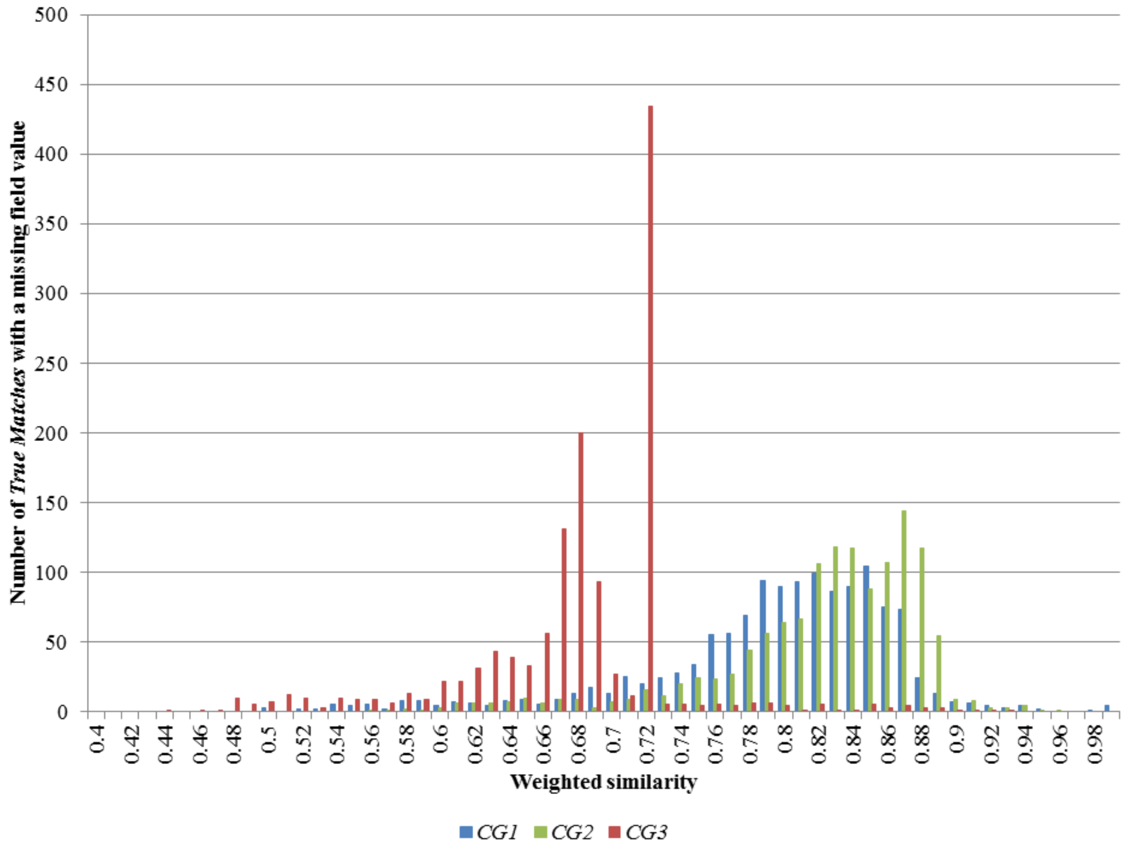


Figure 6.7: The similarity of *True Matches* with a missing field value.

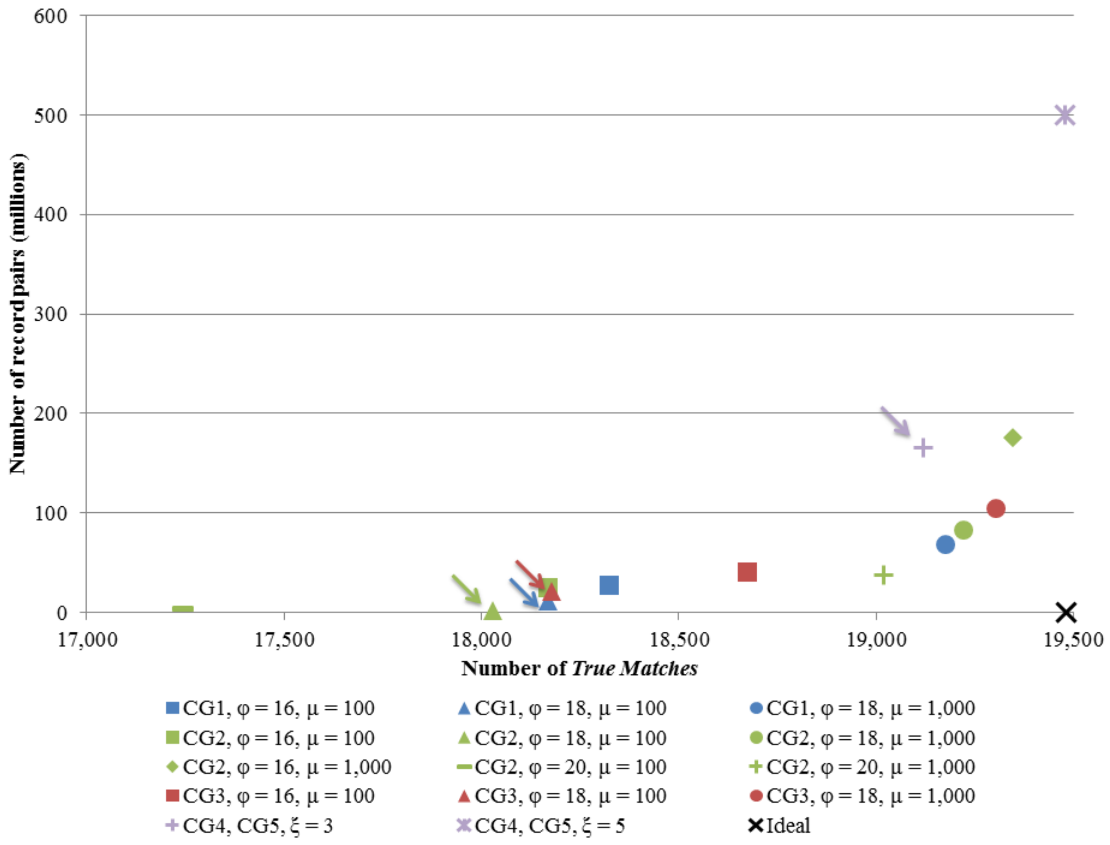


Figure 6.8: The number of record pairs output for various parameter settings and blocking algorithms. Each method is indicated by a different color, and various parameter settings are indicated with the same symbol across methods.

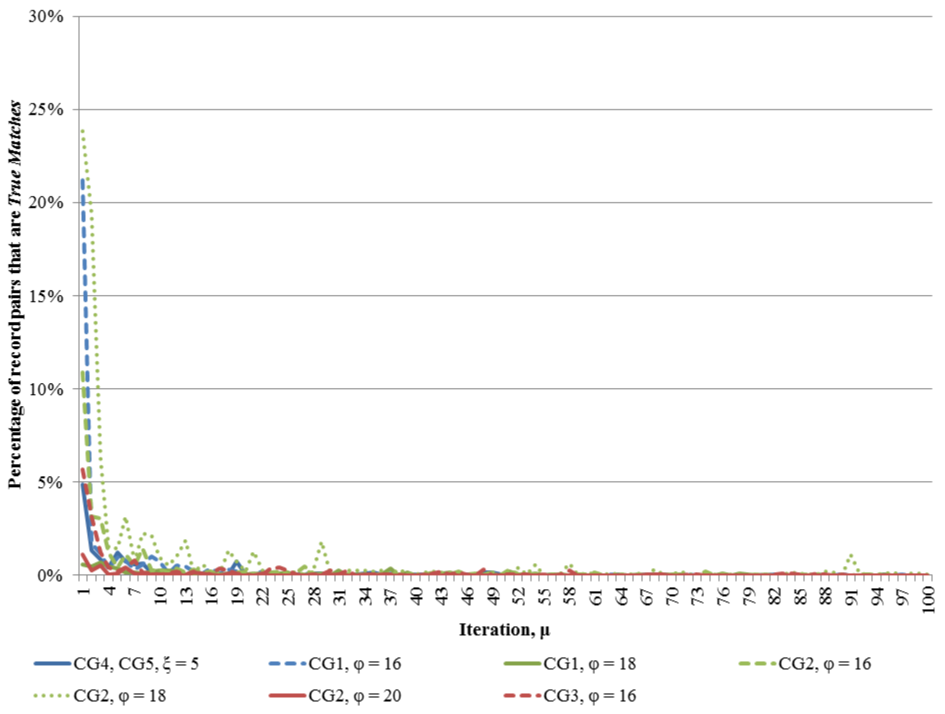
Matches amongst those output record pairs, are plotted in Figure 6.8. The parameter settings selected and used for the remainder of the evaluation are indicated by colored arrows. In all cases, the settings $\phi = 18$ and $\mu = 100$ are applied. The ideal blocking algorithm, indicated with a black X in Figure 6.8, would return all of the *True Matches*, and only *True Matches*, and corresponds to perfect *Precision* and *Recall*.

Note that the parameter settings selected and used for the remainder of the evaluation are shown with colored arrows in Figure 6.8. While alternate parameter settings could be used to retain a greater number of true positives, we believe that these true positives are actually not very similar. If they were, they would have been identified in a previous iteration of the partitioning algorithm. Therefore, they will likely be misclassified in the record pair classification step. We believe that incurring significant additional computational expense to retain these *True Matches* does not seem productive. Consider the evidence presented in Figure 6.9 that supports this hypothesis. This figure shows the percentage of record pairs that are *True Matches*, broken down by iteration. In iteration 1, a significant number of the record pairs output are *True Matches*. However, as the iterations progress, the percentage of record pairs that are *True Matches* drops dramatically. Also, note that this figure only shows iterations 1 through 100 because the percentage of record pairs that are *True Matches* decreases to negligible levels in further iterations.

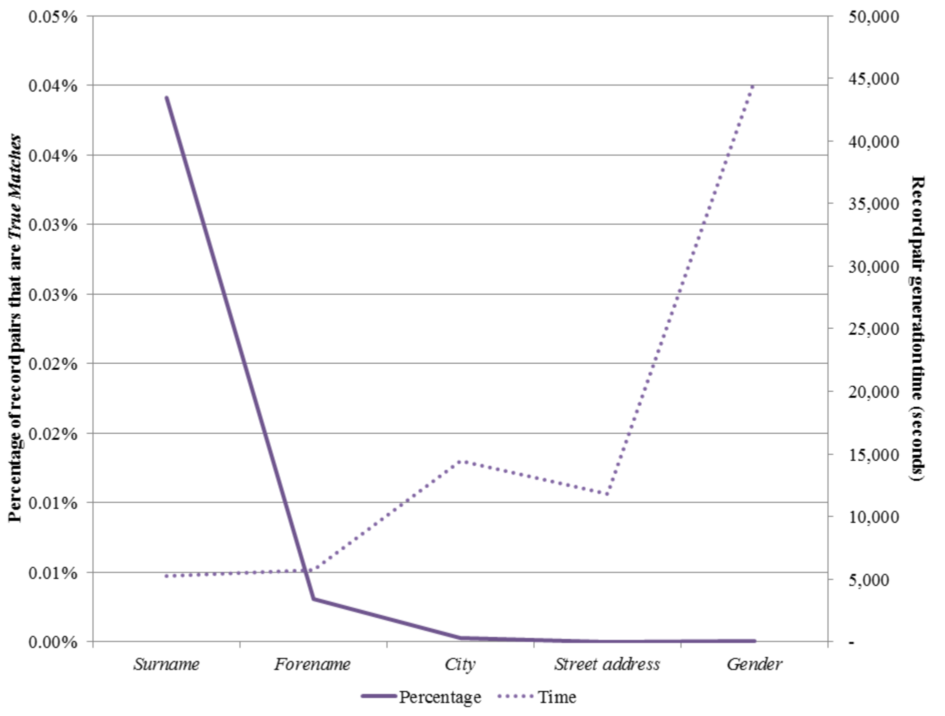
In all HLSH methods, we propose sampling 18 bits per iteration ($\phi = 18$) and iterating 100 times ($\mu = 100$). We explore these parameters settings by examining the similarity of false negatives – *True Matches* not retained by a blocking algorithm – under the parameter settings. The idea is that if false negatives are very similar, the parameter settings should be adjusted to better identify these *True Matches*. However, if the false negatives are dissimilar, they are unlikely to be correctly classified by any blocking algorithm. The average similarity of false negatives for these parameter settings under *CG1*, *CG2*, and *CG3* are 74.6% ± 6.2%, 75.9% ± 5.4%, and 75.9% ± 6.7%, respectively. This provides further support for the hypothesis that the *True Matches* not identified under these parameter settings are not very similar and are, therefore, likely to be misclassified in later stages of the linkage process.

Figure 6.9 shows the percentage of output record pairs that are *True Matches*, broken down by iteration for the *Token* blocking algorithm. Figure 6.9(b) also shows the running time for the *Token* blocking algorithm, which emphasizes that, as the iterations continue, not only do the number of output record comparisons increase, but the running time of the blocking algorithm also increases. The information shown in this figure supports the exclusion of the fields *City* and *Gender* from use as a blocking variable. Therefore, we use $\nu = 3$ for the Token blocking algorithm, and retain the fields *Surname*, *Forename*, and *Street address*.

The running times for all algorithms and parameter settings are shown in Figure



(a) HLSH methods.



(b) Token method.

Figure 6.9: Breakdown of the blocking results by iteration with the Vanderbilt dataset. Each field represents a single iteration, ν, of the token blocking algorithm.

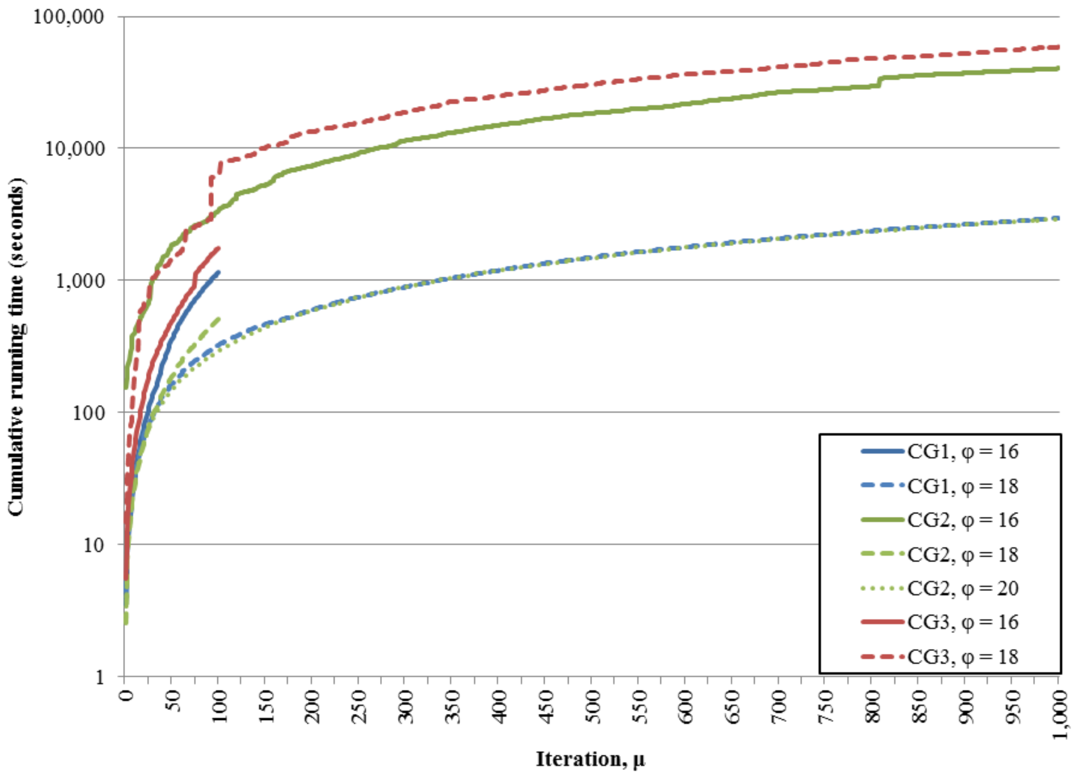
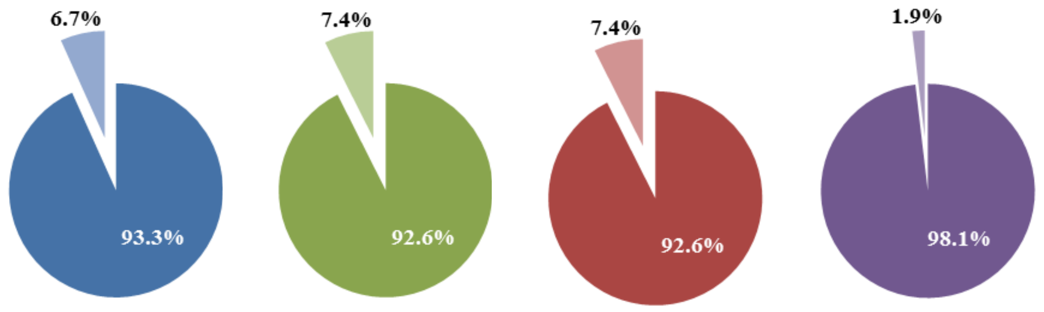


Figure 6.10: Cumulative running times for blocking algorithms on the Vanderbilt dataset.

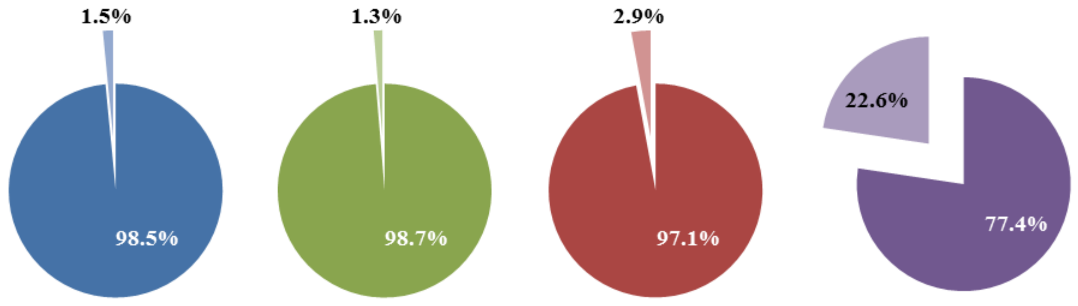
6.10. When it was obvious that 1,000 iterations would yield unreasonable running times and number of output record pairs (as was frequently the case when $\phi = 16$), the evaluation was halted at $\mu = 100$. As μ increases, the running time also increases. A general trend is that as ϕ increases, the running time decreases. However, *CG3* is an exception to this trend for the reasons discussed in Section 6.3.1.

The PP and RR values for the selected blocking methods and parameter settings for methods *CG1* through *CG4* are shown in Figures 6.11 and 6.12, respectively. The experimental HLSH and m-HLSH methods identify approximately 93% of *True Matches* while reducing the number of record pair comparisons by more than 97%. The control *Token* method identifies over 98% of *True Matches*, but reduces the number of record pair comparisons by only 77.4%.



(a) *CG1*: HLSH, ϕ = 18, $\mu = 100$. (b) *CG2*: HLSH, ϕ = 18, $\mu = 100$. (c) *CG3*: m-HLSH, ϕ = 18, $\mu = 100$. (d) *CG4*,*CG5*: Token, ν = 3.

Figure 6.11: The darker color corresponds to *Pairs Completeness*, and the lighter color corresponds to the percentage of *True Matches* that were not identified for further evaluation.



(a) *CG1*: HLSH, ϕ = 18, $\mu = 100$. (b) *CG2*: HLSH, ϕ = 18, $\mu = 100$. (c) *CG3*: m-HLSH, ϕ = 18, $\mu = 100$. (d) *CG4*,*CG5*: Token, $\nu =$ 3.

Figure 6.12: The darker color corresponds to the *Reduction Ratio*, and the lighter value corresponds to the percentage of possible record pair comparisons that are required under a given blocking algorithm.

6.4.3 Linkage Results

Following the blocking step, record pairs are compared to one another and then classified into the set of *Predicted Matches* and *Predicted Non-matches*. Figure 6.13 shows the *Precision*, *Recall*, and the *F-score*, as defined in Equation 5.3, of *Precision* and *Recall* for methods *CG1* through *CG5*, broken down by method for record pair classification. The *Reciprocity* classification always outperforms the other classification methods in terms of *F-score*. Binary does not do a good job of picking up *True Matches*, as indicated by the low *Recall* values. The highest *F-score*, 0.89, is achieved by *CG4* , followed by *CG2* at 0.88. *CG3* has a *F-score* of 0.87 while *CG3* and *CG5* have a *F-score* of 0.86.

6.5 Discussion

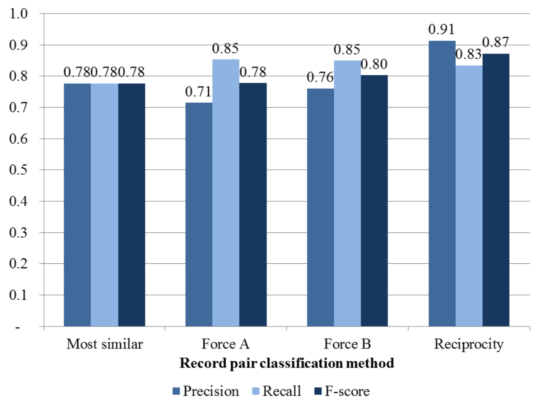
We now discuss the results with respect to each step of the record linkage process.

6.5.1 Field Selection

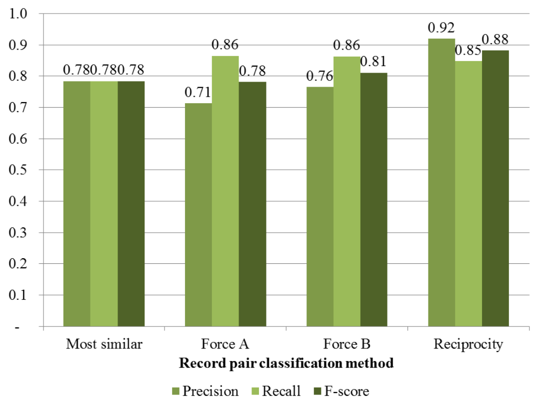
In field selection, a set of fields is identified as input to record linkage. In this work, we choose five fields for inclusion in record linkage (*Surname*, *Forename*, *City*, *Street address*, and *Gender*), and ensured that all individuals were unique given those fields. *Ethnicity* was not included because over 50% of records were missing the *Ethnicity* value.

6.5.2 Field Weighting

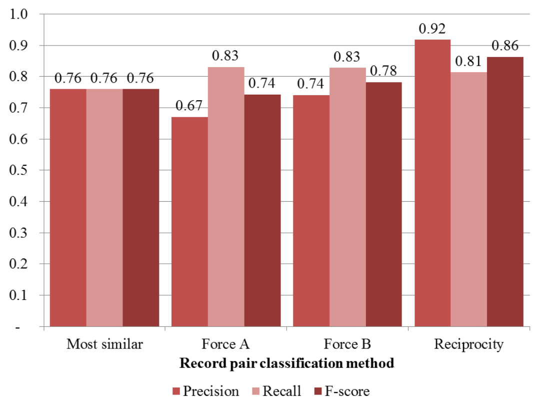
In this dissertation, we have focused on the FS algorithm for field weighting (described in Section 2.1.6). This requires either a subset of records for which the true match status is known, or a method to estimate the conditional probabilities of field agreement given match status. In this chapter, we assume that a subset of records for which the true match status is known, is available. Specifically, we use a subset of 1,000 records from datasets A and B such that each record in the subset drawn from A has a *True Match* in the subset drawn from B. We calculate the γ similarity vectors for these record pairs, and used these to



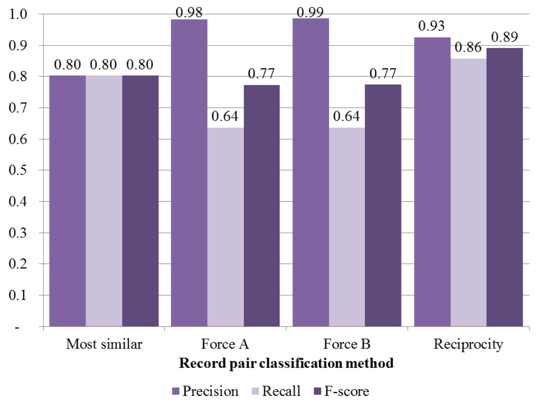
(a) *CG1*.



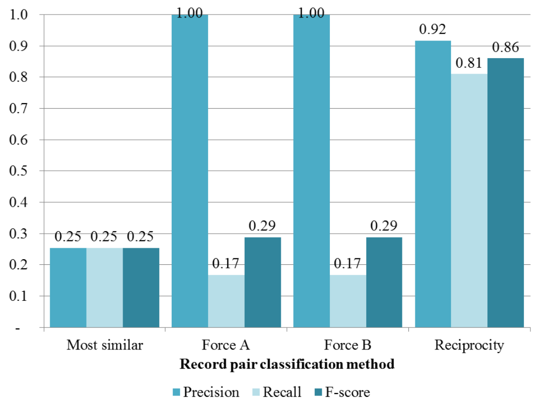
(b) *CG2*.



(c) *CG3*.



(d) *CG4*.



(e) *CG5*.

Figure 6.13: The *Precision*, *Recall*, and *F-score* of these values, for *CG1* through *CG5*.

calculate the conditional probabilities of field agreement given match status.

We test two methods for field weighting in the context of missing field values: 1) *Treat missing as disagreement* (used in *CG1* and *CG2*), and 2) *Exclude missing and discount* (used in *CG3*, *CG4*, and *CG5*). The field weights associated with both methods are shown in Table 6.2. Since so few records in this dataset have missing values (see Table 6.5), there was little difference in the field weights output by the different methods.

In the subsection to follow, we discuss how the PRL framework can be modified to provide for privacy-preserving field weighting.

PRL Framework that Provides for Field Weighting

Recall that Alice and Bob are the data holders linking their records. Charlie is a third party who receives encoded data from Alice and Bob and performs the linkage. In order for Alice and Bob to calculate the field weights associated with their datasets, we propose the use of an additional third party, who we call Dan. A protocol for PRL based on RBFs encodings created by weighted sampling of bits from FBFs encodings involves the following steps:

1. Alice and Bob agree upon data standardization conventions, a hash function, and a key to use.

2. Alice concatenates the key to each field value in each of her records and hashes each field value and sends it to Dan.

3. Bob concatenates the key to each field value in each of her records and hashes each field value and sends it to Dan.

4. Dan compares the record pairs he has received and uses the resulting similarity values as input to the Expectation Maximization algorithm.

5. Dan sends the results of the EM algorithm to Alice and Bob.

6. Alice and Bob agree upon n, q, m, k, the hash functions and keys (*i.e.*, salts) to use in generating FBF encodings, a sample of bits to pull from FBFs (that is weighted

based on the field weights received from Dan) to create the RBFs, and a random permutation of the bits in the RBF.

7. Alice encodes her data and sends it to Charlie.

8. Bob encodes his data and sends it to Charlie.

9. Charlie optionally performs blocking on the RBF encodings.

10. Charlie compares the RBF encodings using the Dice coefficient.

11. Charlie classifies record pairs as either a *Predicted Match* or *Predicted Non-match*.

12. Charlie reports the results of the linkage to Alice and Bob.

This protocol differs from the one used throughout this work (see Section 2.2) in that it requires an additional third party, Dan. Dan provides a means for Alice and Bob to determine field weights relevant to their datasets to use in creating encodings. The reason that Charlie should not play this role is that since he will see the resulting encodings, he should not be privy to the information used in the encoding process.

6.5.3 Encoding

RBF encodings are generated by dynamically sizing FBFs, based on expected field length, and sampling from the FBF encodings according to field weights. In this work, we use a security constraint $q = 4$, such that only bits that can be mapped back to four FBFs, by examining the frequency at which they are set, can be included in the RBF. When missing values are encoded as '0', 606 of the FBF bits satisfy this constraint, but 337 bits are excluded from inclusion in the RBF. These FBFs are used to create the RBFs for methods *CG1* and *CG3*. When missing values are set randomly, 598 of the FBF bits satisfy this constraint, but 345 bits are excluded from inclusion in the RBF. These FBFs are used to create the RBFs for method *CG2*. Additional details can be found in Figures 6.4 and 6.5.

In this chapter, we explore three methods for encoding missing field values in Bloom filter encodings:

1. *Missing bits are encoded as '0'.* The bits corresponding to missing field values remain at the default value of 0. This method is used in method *CG1*.

2. *Fill in missing bits at the expected frequency.* The bits corresponding to missing field values are randomly set with probability p, where p is the average probability with which bits are expected to be set to dynamically size FBFs. In this dissertation, we let $p = 0.5$. This method is used in method *CG2*.

3. *Missing bits are explicitly denoted.* The bits corresponding to missing field values are explicitly denoted with a non-binary character and a weighted form of the Dice coefficient is used such that these bits are not included in similarity calculations (see Equation 6.1). This method is used in method *CG3*.

In terms of security, *CG2* is the most secure, followed by *CG1*, then *CG3*. Recall that one major benefit a RBF encoding provides is that it makes it more difficult for an attacker to determine to which field a given bit corresponds. When the bits corresponding to a missing field value are explicitly denoted in *CG3*, an attacker can see the non-binary value indicating a missing field value, and use this information to map RBF bits back to fields. Similarly, when bits corresponding to a missing field value are represented by '0' in *CG1*, an attacker can observe that certain bits are not set at the expected frequency p, and can use this information to map RBF bits back to fields. However, when the bits corresponding to missing field values are set with the background probability p in *CG2*, an attacker does not gain any information to aid in mapping RBF bits back to fields.

Additionally, the empirical results suggest that the encodings produced under method *CG2* result in encodings more reflective of true similarity, thus facilitating more accurate linkage results. Specifically, *CG2* better separates *True Matches* and *True Non-matches* (see Figure 6.6), *True Matches* appear more similar (see Figure 6.7), and results in more accurate linkage predictions (see Figure 6.13). By contrast, explicitly indicating missing field values and calculating a weighted similarity (see Equation 6.1), under method *CG3*, results in encodings that tend to underestimate the true similarity, thus facilitating less accurate linkage results. Encoding missing data fields with '0' in Bloom filter encodings, as done in method *CG1*, results in a performance intermediate to *CG2* and *CG3*.

6.5.4 Blocking

In the blocking step, records are partitioned, and only records within each partition are compared to one another, with the goal of reducing the number of record pair comparisons required. HLSH is used in methods *CG1* and *CG2* while a variation, m-HLSH, is used when bits corresponding to missing field values are explicitly denoted in method *CG3* (see Figure 6.3). The idea behind the m-HLSH approach is that if partition decisions are based on bits corresponding to present (*i.e.*, not missing) field values, fewer *True Matches* will be incorrectly excluded from further evaluation. For all HLSH methods, we test various parameter settings for ϕ, the number of bits sampled, and μ, the number of iterations performed. A token-based blocking approach is used for methods *CG3* and *CG4* such that if the first letter of a field agrees, the record pairs are placed in the same partition. We let ν refer to the number of fields used for this approach.

Figure 6.8 shows the number of record pairs output for each method and parameter setting plotted against the number of *True Matches* output. When m-HLSH is used, more of the *True Matches* are retained, as hypothesized, but at the expense of additional record pair comparisons. Other general trends are that for a given value of μ, as ϕ decreases, the number of *True Matches* output increases. However, for a given ϕ value, as μ increases, the number of *True Matches* output increases.

For the token blocking approach, we consider values of ν in $\{3, 5\}$. When $\nu = 3$, we partition based on the first letters of *Surname*, *Forename*, and *Street address*. If the first letter of any of these field values agree, the record pair will be output from this blocking algorithm. When $\nu = 5$, we also partition based on the first letter of *City* and *Gender*. Since this is a geographically localized dataset, many of the represented individuals live in the same city. Therefore, partitioning on this field does not result in a significantly reduced number of record pair comparisons required. Similarly, because the domain of the field *Gender* is so small, partitioning based on this field does not result in significant savings. While including these fields slightly increases the number of *True Matches* output (by approximately 400), it comes at significant computational cost, both in terms of the additional number of record pair comparisons output ($>$ 300 million additional record pair comparisons), but also in terms of the time required for the blocking algorithm itself ($>$ 16 additional hours to partition based on these two fields). This is highlighted in Figure 6.9(b).

In selecting a parameter setting for each method to carry forward in the evaluation, we consider the "return on investment" afforded by each iteration μ. Figure 6.9 shows the percentage of record pairs output at each iteration that are *True Matches*. When $\phi = 1$, the percentage of record pairs that are *True Matches* is significant (>20% in some cases). As ϕ increases, this percentage drops, and while you still pick up record pairs at each iteration, very few of these are *True Matches*. This observation raises a question about the goal of blocking, and the answer is user-specific based on his resources and desires. If the goal of blocking is to identify, with high probability, *similar* record pairs, then halting at $\mu = 100$ seems sufficient. However, if the goal of blocking is to identify, with high probability, *True Matches*, then using increased values of μ is more in line with this goal, pending the user is willing to accept the additional computational burden required to identify these dissimilar *True Matches*.

In this chapter, we select $\phi = 18$ and $\mu = 100$ for *CG1*, *CG2*, and *CG3*. The idea behind this decision is that *True Matches* that don't agree on 1.8% of their bits 100 times in a row are unlikely to be similar. If the *True Matches* are dissimilar, they are likely to be misclassified as *Predicted Non-matches* in later steps of the linkage. Therefore, we do not believe the significant additional computational effort to retain these dissimilar *True Matches* in the blocking step is justified.

6.5.5 Record Pair Classification

We consider four different ways to classify record pairs into the sets of *Predicted Matches* and *Predicted Non-matches*, given a set of record pair similarities: 1) *Most similar*, 2) *Force A*, 3) *Force B*, and 4) *Reciprocity*. Further details on these methods can be found in Section 6.3.1. Figure 6.13 shows the *Precision*, *Recall*, and the *F-score* of these values for all methods. In general, the *Reciprocity* classification method outperforms the other classification methods.

One drawback associated with the *Most similar* method of classification is that it allows for a single record in dataset A to be linked to multiple records in dataset B, and vice versa. However, if both datasets A and B are de-duplicated (see Section 2.1.1), this should not be the case. For example, in method *CG1*, when the *Most similar* classified method is used, 1,989 records in dataset A are involved in more than one link prediction, and 2,171 records in dataset B are involved in more than one link prediction.

6.5.6 Overall Linkage Results

If we consider only the results of the *Reciprocity* classification method in Figure 6.13, we see that *CG4* has the highest *F-score* (0.89), followed by *CG2* (0.88). One reason that *CG2* may not perform as well is that we have enforced a stringent security constraint $q = 4$. This means that many bits (345 in the case of *CG2*) from FBFs are not eligible for inclusion in the RBF. This increased security may come with a slight cost to accuracy.

One reason that all methods do not identify more *True Matches* (the highest *Recall* value is 0.86 for *CG4*, *Reciprocity*) is that this is a very challenging dataset. 1,402 records have a missing field value (see Table 6.5 for details), and of the 19,482 *True Matches*, only 3,051 agree on all fields. Also, we believe that the context in which this dataset was generated (*i.e.*, for clinical care) leads to some additional properties that make record linkage more difficult. For example, it may have been more useful to use a name when referring to a patient, even if the name was not accurate. Examination of the dataset leads us to hypothesize that when fields are missing, sometimes this data has been filled in with inaccurate information. For example, several individuals are "John Doe". While we attempted to identify and remove data that is not meaningful, this was not always possible.

6.6 Conclusions

This chapter presents a PRL framework that provides for efficient, accurate, and secure linkage. We believe method *CG2*, based on RBF encodings, where missing field values are randomly set, offers the best performance in terms of accuracy and security.

6.6.1 Future Work

In Chapter 7, we summarize the work presented in this dissertation, discuss its implications, and highlight open research questions.

CHAPTER VII

CONCLUSIONS AND FUTURE WORK

In this chapter, we summarize the work presented in previous chapters, reflect upon its implications and limitations, and discuss open research questions.

7.1 Summary of Chapters

Chapter 1 defined record linkage and its privacy-preserving instantiation, PRL. PRL has broad implications across multiple domains, but is particularly useful in the field of medicine, where regulations governing health data limit the disclosure of identifying patient data.

Chapter 2 provided an overview of the steps involved in record linkage and additional background information.

Chapter 3 presented the results of a survey and evaluation of existing methods for privately comparing field values. The encoding methods were evaluated in terms of their correctness, efficiency, and security. An encoding technique, based on the use of Bloom filters, was found to be both highly accurate and efficient. This portion of the research has resulted in several publications [41, 42].

Chapter 4 presented a novel method for encoding field values in Bloom filters that addresses security vulnerabilities identified in existing Bloom filter encoding techniques. Briefly, rather than encoding each field value in a distinct Bloom filter, a single RBF is used. This makes it more difficult for an attacker to leverage field-level frequency distributions to determine information about the plaintext values from which the encoded values were derived. Additionally, a method to dynamically size Bloom filters, based on the expected length of field values, further impedes the extent to which an attacker can leverage frequency information. Finally, a security constraint, q, was introduced that users can tune to provide security guarantees based on their requirements. We show that this novel encoding technique is distance-preserving and provides for accurate comparisons of the encoded values.

Chapter 5 laid out a private blocking algorithm that paves the way for a rapid, yet accurate, record linkage application. This algorithm is based on the application of LSH functions that approximate Hamming distance, to the RBF data structure developed in Chapter 4. This algorithm was compared to several controls and both the extent to which it reduces computational complexity and preserves accuracy were evaluated. This algorithm was found to be superior to controls in both categories.

Finally, Chapter 6 brings together the entire PRL framework and presents an evaluation using the Vanderbilt dataset that addresses how to handle missing values.

7.2 Lessons Learned

We now highlight some of the lessons, as they relate to each step of the record linkage process, learned throughout this research.

7.2.1 Field Selection

In field selection, a set of fields is identified as input to record linkage. Field selection is related to the task of feature selection, a first step in machine learning algorithms, and can significantly affect algorithm accuracy [81]. It is imperative that the set of field values chosen be sufficient to uniquely identify an individual. Otherwise, the linkage predictions could not be meaningful and could not support the claim "Since these records agree on all fields, they likely refer to the same individual".

When considering fields for inclusion, fields with the following properties are best suited for record linkage:

1. **Highly discriminatory.** Fields that are highly discriminatory in their ability to distinguish between individuals are ideal for record linkage.

2. **Accurately recorded.** Fields that are likely to contain errors are not preferred.

3. **Consistently recorded.** Fields that are often missing introduce complications.

4. **Stable over time.** Field values that are likely to remain unchanged over time are preferred over fields that may change over time, such as *Surname*.

5. **Drawn from the same domain.** With respect to security, fields that are drawn for the same domain (*i.e.*, all string-based fields, all categorically-valued fields, *etc.*) are more difficult to differentiate between when encoded in a RBF (please see Section 4.5.2 for further details).

Records that lack sufficient information to uniquely identify an individual should not be input to a linkage algorithm. In addition to being unable to correctly classify the record, these records may throw off parameterization methods based on the set of input records, and thus adversely affect the linkage exercise as a whole.

Rather than creating a record linkage framework tolerant of missing field values, an alternative approach is to perform iterative linkages where records not linked in initial linkages are held over to latter linkages that are performed on a subset of the fields. For example, if five fields are available for linkage, the records where all five fields are present are input to the initial linkage. Records with missing field values are set aside. A second linkage is then performed using four fields. Records where these four fields are present, and records not linked in the initial linkage, form the input to the second linkage. The iterations should halt when a set of fields that are not sufficient for uniquely identifying an individual are reached. A complication associated with this method is that it can be computationally complex, and the order in which the linkages should be performed is unclear.

7.2.2 Field Weighting

In this dissertation, we have focused on the FS algorithm for field weighting (described in Section 2.1.6). This requires either a subset of records for which the true match status is known, or a method to estimate the conditional probabilities of field agreement given match status.

Practical Challenges Associated with Field Weighting

While an EM algorithm can be used to estimate these conditional probabilities [49, 66, 108, 118], the estimates are highly dependent upon the quality of the input dataset. Estimates in the face of missing field values are particularly unreliable [14].

Additionally, just determining which information to input to EM can be a challenge. For instance, EM accepts as input a set of binary γ similarity vectors. Since the similarity vectors associated with all record pairs can generally not be computed efficiently (hence, the need for blocking algorithms), a subset of record pairs must be used instead. A random sampling of record pairs is unlikely to yield a sufficient number of *True Matches* for reliable parameter estimation [119] because the number of *True Non-matches* significantly dominates the number of *True Matches*. Therefore, a method for enriching *True Matches* is required. One approach to do so is to use a blocking method to select a subset of record pairs likely to contain *True Matches*. However, the sample resulting from the blocking technique is not necessarily representative of the entire dataset. We hypothesize that using a *True Match* enrichment technique similar to the blocking technique employed in record linkage will yield estimates relevant for the linkage.

Even when representative inputs are available, EM is not guaranteed to return accurate parameter estimates. EM is a maximum likelihood estimation algorithm that seeks to select the most likely parameter set given the inputs. In the case of record linkage, the parameters estimated are the conditional probabilities of field agreement given match status, and the inputs are the γ vectors. However, in searching the landscape of possible parameters, the algorithm is not guaranteed to select the globally optimal parameter set. Several variables that affect how the algorithm performs are the convergence criteria used and the initial seed values for parameter estimates.

In summary, while EM can theoretically be used to estimate field weights, there are some challenges associated with its implementation in practice. Additionally, it does always yield reliable parameter estimates, particularly in the face of missing data. Field weighting using the EM algorithm is an aspect of the record linkage process not addressed in this dissertation and is left to future work.

7.2.3 Encoding

We propose RBF encodings that are generated by dynamically sizing FBFs, based on expected field length, and sampling from the FBF encodings according to field weights. We introduce a security constraint q, such that only bits that can be mapped back to $\geq q$ FBFs, by examining the frequency at which they are set, can be included in the RBF.

Missing field values must be carefully handled in encoding so that they do not reveal information an attacker can leverage. We found that randomly setting the bits, with background probability p, corresponding to missing field values, in the context of Bloom filter encodings, provides the best balance of security and accuracy.

7.2.4 Blocking

In the blocking step, records are partitioned and only records within each partition are compared to one another, with the goal of reducing the number of record pair comparisons required. Generally, blocking presents a tradeoff between accuracy and computational complexity – when more record pairs are identified for further evaluation, more *True Matches* are retained; when fewer record pairs are identified for further evaluation, fewer *True Matches* are retained.

Users must decide at which point on this scale between computational complexity and accuracy they prefer based on their computational resources and their desired level of accuracy, and set blocking parameters accordingly. This principle applies for all blocking algorithms, but the specific parameters relevant to the HLSH algorithm presented in this work are ϕ, the number of bits selected from the RBF to partition the records, and μ, the number of iterations applied. Computational complexity and accuracy have an inverse relationship with the parameter ϕ and a direct relationship with the parameter μ.

7.2.5 Record Pair Classification

In this work, we presented four heuristics for record pair classification (see Section 6.3.1). However, the most appropriate method for record pair classification is dependent upon the application and desired results in practice. For example, if *Precision* is more important than *Recall* for a particular record linkage application, that must be taken into account in record pair classification.

One drawback associated with some record pair classification methods (*i.e.*, the *Most similar* method of classification used in Chapter 6 or the application of a single threshold where record pairs having a similarity greater than this threshold value are classified as *Predicted Matches*) is that a single record in dataset A may be predicted to link to multiple records in dataset B, and vice versa. However, if both datasets A and B are de-duplicated (see Section 2.1.1), this should not be the case. This results in a problem similar to the stable marriage problem [52], which must then be addressed by applying an algorithm to ensure each record in dataset A is linked to at most one record in dataset B (and vice versa) in an optimal way. In practice, this may be enforced through an exhaustive linear sum assignment procedure or a greedy matching heuristic. Further details can be found in the work by Len *et al.* [78].

Fellegi and Sunter suggest a method for record pair classification based on an estimate of the error rates associated with given boundaries [46]. An alternative method for classifying record pairs is based on an extension to the FS algorithm [77] to estimate $P((a, b) \in M|\gamma(a, b))$. One challenge associated with this method, in practice, is determination of this probability. This becomes an even greater challenge when approximate measures of field similarity are allowed, because the size of the problem increases. Other machine learning algorithms, such as support vector machines [30], artificial neural networks [16], and clustering algorithms [19], could also be considered for classifying record pairs.

7.3 Limitations and Open Research Questions

We now discuss limitations of this work and several open research questions that are left to future work.

This work utilizes two datasets. The first is composed of publicly available records drawn from the NCVR dataset and a corrupted version of these records. While errors are introduced at rates consistent with those expected in practice, this dataset is semi-synthetic, and the extent to which the results will generalize to other datasets is unclear. The second dataset consists of demographic information drawn from patient records at Vanderbilt University. A significant number of field values are missing from records in this dataset, and therefore this dataset is a "difficult" case, and we anticipate that the methods presented will result in better performance on more complete, clean datasets. The semantics of the data may have affected the results, and the parameters may not generalize to other datasets, but we consider this work a demonstration of the feasibility of the approach.

Additionally, multiple computational resources were used throughout this work. While the use of multiple computers affects the specific running times reported, it does not influence the findings and conclusions of this study.

In the remainder of this section, we discuss practical challenges to implementing PRL and extending PRL to other scenarios.

7.3.1 Practical Challenges to Implementation

In this section, we discuss issues that need to be addressed for a PRL implementation to be realized in practice. The protocol used throughout this dissertation utilizes a third party, Charlie, to perform the record linkage. Charlie should be a trusted entity, and data sharing agreements should be used to supplement the security properties of encoded data.

Also, Alice and Bob need to agree on some information. Specifically, they need to agree upon the fields to use, data standardization practices and pre-processing rules, the information required to generate FBF encodings (*i.e.*, n, m_{FBF} for each field, k, which hash functions to use, and salts to apply in hashing) and the information required to generate RBF encodings (*i.e.*, the security constraint q, field weights, which bits to include in the RBF, and a random permutation of bits in the RBF). In practice, Alice and Bob could talk directly to one another to agree upon these items. However, this does not scale if linking across additional data providers is desired, as discussed below. Alternatively, an additional third party can be used to aid in determining and distributing the required information[*]. When parameter determination requires data from Alice and Bob (*i.e.*, field weighting using EM, or determining which bits in FBF encodings satisfy security constraints), an additional third party, Dan, can be utilized, as discussed in Section 6.5.2.

A work recently published by Weber *et al.* [116] essentially notes that while the approach is precise, the lack of a publicly available software toolkit is a barrier to its implementation:

> *"Other approaches [such as] Durham* et al. *are far more precise... Note that the complexities of [the] Bloom filter approaches require retaining the services of highly specialized programmers to implement, which is why we did not pursue either of these options for our study."*

To address this barrier, an extension to the Open Enterprise Master Patient Index software (OpenEMPI) [1] is under development. The software toolkit, called Secure OpenEMPI (SOEMPI) will be open source and publicly available at `http://hiplab.org/SOEMPI`.

7.3.2 Extending PRL

Extending PRL to a Decentralized Setting

If Alice and Bob prefer to perform PRL without using a third party, Charlie, they can use the secure dot product protocol [83] to calculate the intersection of bits in the Bloom filter

[*]However, it is important that Charlie does not play this role. If he is to receive the encoded data, and also has knowledge of the information used in encoding the data, he may be able to use that information to determine the plaintext values.

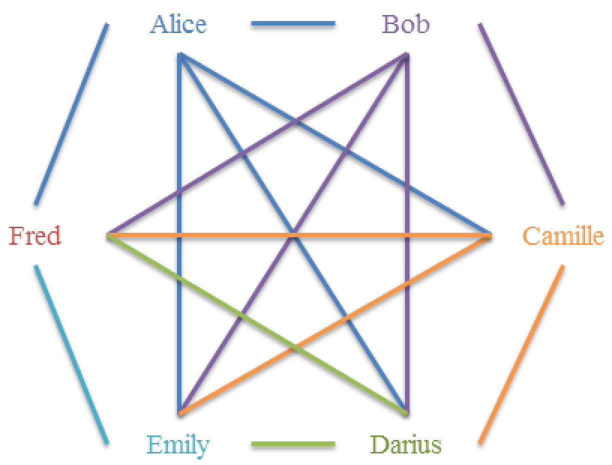


Figure 7.1: The pairwise linkages required to link six datasets.

without requiring sharing of the encoding. They can then determine the number of bits set in total in each of their Bloom filters using additively homomorphic encryption, such as the Paillier cryptosystem [91]. This information is then sufficient for calculating the Dice coefficient and determining the similarity of the record pair without requiring sharing of the encoding or the use of a third party. However, this procedure requires more time than the protocol based on sharing Bloom filter encodings with a trusted third party who calculates the Dice coefficient.

Extending PRL to More than Two Datasets

Throughout this work, we have considered a pairwise linkage of records; specfically that of linking a dataset A, held by Alice, to another dataset B, held by Bob. However, in practice, data holders Alice, Bob, Camille, Darius, Emily, and Fred might like to link all of their datasets. If a pairwise linkage approach is used, in this example involving six data holders, 15 linkages must be performed as shown in Figure 7.1[†]. In general, the number of linkages that must be performed is $\sum_{☼=1}^{\xi} ☼ - 1$ where ξ is the number of data holders. As ξ grows, this approach can become laborious and impractical.

One alternative is to create a centralized data repository to which data holders can iteratively link their datasets. This raises several questions that must be answered in

[†]This assumes the linkage relationship is symmetrical (*i.e.* , Link(A,B) is equivalent to Link(B,A))

practice:

1. How should the initial repository be created?

2. In what order should other data holders link to the centralized repository and how does it affect the results?

3. When links are identified, how does this change the information stored for an individual?

Recall that the field weights used in RBF generation, as described in Section 4.3.1, are specific to the datasets of Alice of Bob. One benefit to determining field weights in this manner is that the values will be representative of the discriminatory power of the fields in the datasets to be linked, and will therefore facilitate an accurate linkage of these datasets. However, the extent to which those field weights generalize to other datasets is unclear. For example, in the case of the NCVR dataset, the *State* field has the same value for all records and therefore essentially no discriminatory power in resolving identity. However, if the dataset contained records from many states, the field *State* may have greater discriminatory power. Therefore, when linking across multiple data providers, a global set of field weights, that are applicable to all datasets involved, should be used. However, the scalability this affords may come at a cost of decreased accuracy.

Extending PRL to More than Two Datasets in a Decentralized Setting

Finally, we consider the extension of PRL to multiple datasets in a decentralized setting. This is related to entity resolution [17], a task similar to record linkage, but differing in several ways in practice [2]. First, entity resolution involves pulling together the records referring to an individual in real time and is referred to as an “online” process. This in contrast to the one-off, “offline” linkage of two datasets. Second, record linkage is usually performed in batch mode (i.e., all records in datasets A and B are linked), whereas entity resolution is typically used to identify all records referring to a specific entity of interest.

An application for entity resolution in clinical care is identification of a patient's records held by other care providers. In the context of the PRL framework presented in this dissertation, one idea that could be explored is the use of commutative hash functions to generate RBF encodings that can be used to link across multiple datasets in a de-centralized way.

7.4 Implications for Policies Governing the Use of Medical Data

Finally, we recognize the importance of the policies that govern application of the technology described in this dissertation. In this section, we discuss this work as it relates to the policies governing medical data, with a particular focus on the U.S. health care system. Two key questions are of interest:

1. **Does encoded data satisfy the requirements for de-identified data?** Since HIPAA only covers identified data, if encoded data satifies the requirements for de-identification, encoded data can be shared for PRL.

2. **Is an RBF encoding considered an identifying code?** To meet the "Safe Harbor" criteria for de-identification under the HIPAA Privacy Rule, all "identifying codes" must be removed [112]. An identifying code is information derived in from identifying patient information. Since RBF encodings are derived from identifying patient information, it is unclear if they satisfy the Safe Harbor criteria for de-identification.

We begin by considering the policies relevant to the first question. Recently, in the HITECH Act of the ARRA stimulus, it was stated:

"Protected health information (PHI) is rendered unusable, unreadable, or indecipherable to unauthorized individuals if one or more of the following applies:
1. Electronic PHI has been encrypted as specified in the HIPAA Security Rule by the use of an algorithmic process to transform data into a form in which there is a low probability of assigning meaning without use of a confidential process or

key (45 CFR 164.304 definition of encryption) and such confidential process or key that might enable decryption has not been breached. To avoid a breach of the confidential process or key, these decryption tools should be stored on a device or at a location separate from the data they are used to encrypt or decrypt. The encryption processes identified below have been tested by the National Institute of Standards and Technology (NIST) and judged to meet this standard.
(ii) Valid encryption processes for data in motion are those which comply, as appropriate, with NIST Special Publications ... or others which are Federal Information Processing Standards (FIPS) 140-2 validated."

The ARRA statement seems to imply that encrypted data satisfies the requirements of de-identified data, with the caveat that a sufficiently secure encoding function is used ("a low probability of assigning meaning without use of a confidential process or key") and the key is held privately and not breached. The low probability phrasing is ambiguous, but we believe a case can be made that the probability is sufficiently low for protection of PHI.

However, an important stipulation made in the statement is that the key may not be breached. In the context of PRL, Alice and Bob must both use the same key values in encoding their data if the resulting encodings are to be comparable. The following exchange sheds light on the sharing of key values for encoded data and generating encodings derived from patient information. As clarification to the HIPAA Privacy Rule [112], the following comment was made:

"Several commenters who supported the creation of de-identified data for research asked if a keyed-hash message authentication code (HMAC) can be used as a re-identification code even though it is derived from patient information, because it is not intended to re-identify the patient and it is not possible to identify the patient from the code. The commenters stated that use of the keyed-hash message authentication code would be valuable for research, public health and bio-terrorism detection purposes where there is a need to link clinical events on

> *the same person occurring in different health care settings (e.g. to avoid double counting of cases or to observe long-term outcomes)."*

To which, officials in the Federal Government responded:

> *"The HMAC does not meet the conditions for use as a re-identification code for de-identified information. It is derived from individually identified information and it appears the key is shared with or provided by the recipient of the data in order for that recipient to be able to link information about the individual from multiple entities or over time. Since the HMAC allows identification of individuals by the recipient, disclosure of the HMAC violates the Rule. It is not solely the publics access to the key that matters for these purposes; the covered entity may not share the key to the re-identification code with anyone, including the recipient of the data, regardless of whether the intent is to facilitate re-identification or not."*

In the scheme proposed by the commenter, each de-identified record contains a "tag" generated by encrypting an identifier (such as *SSN*) associated with each individual, using a keyed hash function. This keyed hash function is a one-way hash in that given the hash code, there is no mechanism for determining the associated plaintext value[‡]. This essentially creates a unique identifier associated with each individual. However, the tag cannot be used to identify the individual without the presence of a linking table of the form [ID, HMAC(ID)].

The government response states that the HMAC allows identification of individuals by the recipient. However, this is not the case. By definition, given HMAC(key, ID), one is not able to determine ID. The government response also suggests that revelation of the key allows the recipient to determine ID. This is also not explicitly the case. The recipient can

[‡]At this point, we would like to highlight the distinction between hashing and encryption. Encryption is intended to be a "two way" street – an encryption key exists for transforming a plaintext value to a ciphertext value, and a decryption key exists for transforming a ciphertext value to a plaintext value. Hashing, in contrast, is intended to be a "one way" street, where a single key exists to transform plaintext values into ciphertext values

associate an ID with its de-identified record only if they know the plaintext value of the ID. As they already know the key and HMAC(key, ID), they can simply calculate the HMAC value for all of the IDs they have. However, in the general case, given HMAC(key, ID) and key does, a recipient can mount a dictionary attack. In such an attack, the recipient can calculate the HMAC(key, ID) for all possible values of ID, thus creating a linking table of the form described above. When the attacker finds the ID with hash value equivalent to the received hash value, the attacker can link the ID to the de-identified record.

Quantin *et al.* have proposed a scheme involving two pads that prevents such a dictionary attack [94]. In this case, two distinct groups are involved in the protocol – the set of senders and a receiver. Two keys are used: k_1 and k_2. k_1 is static and all senders know its value. k_2 is generated by the receiver and is only known to him. A sender pads the identifier with k_1, hashes it, encrypts it with another form of encryption, such as the Rivest Shamir Adleman cryptosystem [96], and sends the value to the receiver. The reason the padded, hashed identifier is encrypted again is so that it cannot be intercepted by another sender during transmission. If this were to happen, the devious sender could launch a dictionary attack to determine the value of the identifier since they know the key k_1. The receiver then decrypts the value such that they then have the hashed, salted identifier. They then pad this value with k_2 and hash it again. The file can then be released and is not subject to a dictionary attack by senders as the senders only know the value of k_1, but not k_2.

The model proposed by Quantin can work if the model uses a centralized third party who receives records from a group of senders. However, the protocol requires that the sender not reveal the value of k_1 to the receiver. Since a sender cannot be both a sender and a receiver, this model does not work for distributed data exchange amongst a group of hospitals.

However, we believe that as technology is developed, policies can also develop to reflect and support newly enabled applications. The security analyses performed in this work, and in particular discussed in Chapter 4, emphasize the idea that sharing encoded patient data provides comparable privacy protection to the more well-established idea of sharing de-identified patient data[§], pending a sufficiently strong encoding function is used. Such an encoding function has been developed in this research and is described in Chapter 4. Therefore, this technology can be used to inform policy decisions such that its benefits can be realized in practice.

7.5 Conclusions

This dissertation describes a framework for PRL and addresses in detail three crucial aspects of PRL: 1) security, 2) accuracy, 3) and computational complexity. The primary contributions of this work relate to the security and efficiency with which PRL can be performed. Specifically, the RBF encodings described in Chapter 4 provide for approximate field comparison while providing increased security guarantees over FBF encodings. This is achieved by obscuring to which fields bits correspond, thus making a cryptanalysis attack more difficult. Additionally, bits can be sampled from FBFs according to the discriminatory power of the field. The resulting RBF encodings are meaningful representations of the entire record enabling the consolidation of the field and record pair comparisons steps, resulting in time savings. Another major contribution of this dissertation is the application of the HLSH blocking technique to RBF encodings in a way that significantly reduces the computational complexity of record linkage, thus removing a barrier to its implementation. This research provides a framework for secure, accurate, and efficient PRL and paves the way for PRL to be performed in practice to further the field of medicine.

[§]The sharing of patient data in any context should be caveated with the disclaimers that data should only be shared with trusted research or clinical care partners and appropriate data use agreements should govern the transaction.

APPENDIX A

EMBEDDING PARAMETER SELECTION

The *Embedding* comparator [61] evaluated in Chapter 3 of this dissertation uses the Sparse Map variant of Lipschitz embeddings. A detailed description of Sparse Map can be found in [58]. Rather than using all reference sets that compose the embedding space, Sparse Map employs a greedy resampling heuristic to whittle down to a subset containing the most informative strings. These reference sets serve as coordinates that define the embedding space. A recommended number of coordinates was not specified in [61]; therefore, we systematically evaluated all possible subset sizes from 2 coordinates up to the maximum number of coordinates. In this case, the maximum number of coordinates was determined to be 16 (as recommended in [61]). The results are shown in Figure A.1. The highest true positive rate was achieved using subsets of size both 9 and 13. We selected the subset of minimal size and use 9 coordinates to define the embedding space used in this work.

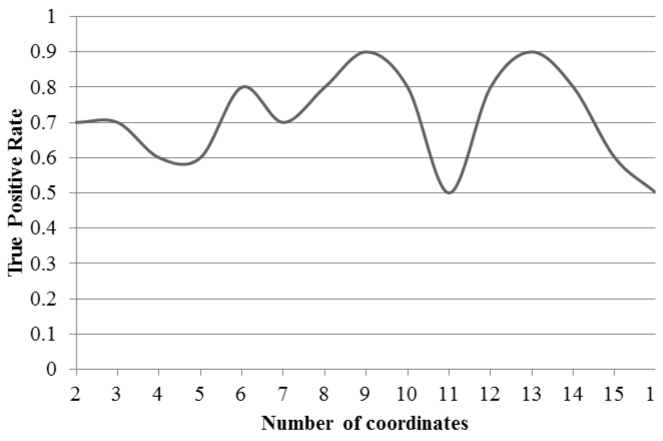


Figure A.1: The TPR achieved for the *Embedding* comparator is dependent upon the number of reference sets used in estimating distance between strings.

CURRICULUM VITAE

Education:-

- **Doctor of Philosophy (Ph.D.)** in **Biomedical Informatics**
  Vanderbilt University, Nashville, TN USA (May 2012)
  Dissertation: "A framework for accurate, efficient private record linkage"

- **Master of Science (M.S.)** in **Biomedical Informatics**
  Vanderbilt University, Nashville, TN USA (August 2008)
  Thesis: "Knowledge-Based environment potentials for protein structure prediction"

- **Bachelor of Science (B.S.)** in **Computer Science**
  Georgia Institute of Technology, Atlanta, GA USA (May 2006)
  Project: Biologically-inspired simulations of computer networks based upon brain networks

Awards & Honors:-

- One of two students from Vanderbilt University nominated to attend a meeting of Nobel Laureates (2010)

- Winner of the American Medical Informatics Student Paper Competition (2010)

- Distinguished Poster Award from the Vanderbilt Institute for Chemical Biology (2008)

- Recipient of the National Library of Medicine Biomedical Informatics Graduate Training Fellowship (2006 - 2010)

Publications:-

1. M. Kuzu, E. Durham, M. Kantarcioglu, and B. Malin, A constraint satisfaction attack on Bloom filters in private record linkage, *Privacy Enhancing Technologies Symposium*, (2011)

2. E. Durham, M. Kantarcioglu, Y. Xue, and B. Malin, Quantifying the correctness, computational complexity, and security of privacy-preserving string comparators for record linkage, *Information Fusion*, (2012)

3. E. Durham, M. Kantarcioglu, Y. Xue, and B. Malin, Private medical record linkage with approximate matching, *Proceedings of the American Medical Informatics Association*, (2010)

4. E. Durham, B. Dorr, N. Woetzel, R. Staritzbichler, and J. Meiler, Solvent accessible surface area approximations for rapid and accurate protein structure prediction, *Journal of Molecular Modeling*, **15** 1093, (2009)

REFERENCES

[1] An open source enterprise master patient index. `http://openempi.kenai.com`, Last accessed 23 March, 2012.

[2] Entity resolution systems vs. match merge/merge purge/list de-duplication systems. `http://jeffjonas.typepad.com/jeff_jonas/2007/09/entity-resoluti.html`, Last accessed 27 February, 2012.

[3] U.S. Food and Drug Administration's Sentinel initiative. `http://www.fda.gov/Safety/FDAsSentinelInitiative/ucm2007250.htm`, Last accessed February 21, 2012.

[4] Massmind nicknames database. `http://techref.massmind.org/techref/ecommerce/nicknames.htm`, Last accessed June 2, 2010.

[5] North Carolina voter registration database. `ftp://www.app.sboe.state.nc.us/data`, Last accessed June 2, 2010.

[6] Abadie, R. The professional guinea pig: Big pharma and the risky world of human subjects. *Australian and New Zealand Journal of Public Health 35*, 3 (2011), 298–298.

[7] Agrawal, R., Asonov, D., and Srikant, R. Enabling sovereign information sharing using web services. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data* (2004), pp. 873–877.

[8] Agrawal, R., Evfimievski, A., and Srikant, R. Information sharing across private databases. In *ACM SIGMOD Interantional Conference on Management of Data* (San Diego, CA, 2003), pp. 86–97.

[9] Aizawa, A. A fast linkage detection scheme for multi-source information integration. In *Web Information Retrieval and Integration* (2005), IEEE Computer Society, pp. 30–39.

[10] Al-Lawati, A., Lee, D., and McDaniel, P. Blocking-aware private record linkage. In *Proceedings of the International Workshop on Information Quality in Information Systems* (Baltimore, MD, 2005), pp. 59–68.

[11] Arellano, M. G., and Weber, G. I. Issues in identification and linkage of patient records across an integrated delivery system. *Journal of Healthcare Information Management 12*, 4 (1998), 43 – 52.

[12] Atallah, M., Kerschbaum, F., and Du, W. Secure and private sequence comparisons. In *Proceedings of the ACM Workshop on Privacy in the Electronic Society* (Washington, DC, 2003), pp. 39–44.

[13] Barlow, T. Your social security number may not be unique to you. `http://www.dailyfinance.com/2010/08/12/your-social-security-number-may-not-be-unique-to-you`, August 2010.

[14] Bauman, Jr, G. J. Computation of weights for probabilistic record linkage using the EM algorithm. Master's thesis, Brigham Young University, August 2006.

[15] Baxter, R., Christen, P., and Churches, T. A comparison of fast blocking methods for record linkage. In *Proceedings of the ACM SIGKDD Workshop on Data Cleaning, Record Linkage and Object Consolidation* (Washington, DC, 2003), pp. 25–27.

[16] Bell, G. B., and Sethi, A. Matching records in a national medical patient index. *Commun. ACM 44* (September 2001), 83–88.

[17] Benjelloun, O., Garcia-Molina, H., Gong, H., Kawai, H., Larson, T. E., Menestrina, D., and Thavisomboon, S. D-swoosh: A family of algorithms for generic, distributed entity resolution. In *Proceedings of the 27th International Conference on Distributed Computing Systems* (Washington, DC, USA, 2007), IEEE Computer Society, p. 37.

[18] Berman, J. J. Zero-check: A zero-knowledge protocol for reconciling patient identities across institutions. *Archives of Pathology & Laboratory Medicine 128*, 3 (2004), 344–346.

[19] Bhattacharya, I., and Getoor, L. Iterative record linkage for cleaning and integration. In *Proceedings of the 9th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery* (New York, NY, USA, 2004), DMKD '04, ACM, pp. 11–18.

[20] Bilenko, M., Basu, S., and Sahami, M. Adaptive product normalization: Using online learning for record linkage in comparison shopping. In *Proceedings of the Fifth IEEE International Conference on Data Mining* (Washington, DC, USA, 2005), ICDM '05, IEEE Computer Society, pp. 58–65.

[21] Bilenko, M., Kamath, B., and Mooney, R. Adaptive blocking: Learning to scale up record linkage. In *Data Mining, 2006. ICDM '06. Sixth International Conference on* (2006), pp. 87 –96.

[22] Bilenko, M., and Mooney, R. J. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining* (New York, NY, USA, 2003), KDD '03, ACM, pp. 39–48.

[23] Binstock, A., and Rex, J. Metaphone: A modern soundex. In *Practical Algorithms for Programmers*, A. Binstock and J. Rex, Eds. Addison Wesley, 1995.

[24] Bloom, B. H. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM 13*, 7 (1970), 422–426.

[25] Broder, A., and Mitzenmacher, M. Network applications of Bloom filters: A survey. *Internet Mathematics 1* (2002), 636–646.

[26] Broder, A. Z. On the resemblance and containment of documents. In *Compression and Complexity of Sequences* (1997), IEEE Computer Society, pp. 21–29.

[27] Buhler, J. Efficient large-scale sequence comparison by locality-sensitive hashing. *Bioinformatics 17*, 5 (2001), 419–428.

[28] Carpenter, P. C., and Chute, C. G. The universal patient identifier: a discussion and proposal. In *Proceedings of the American Medical Informatics Association Annual Symposium* (1993), pp. 49 – 53.

[29] Christen, P. Towards parameter-free blocking for scalable record linkage. Tech. Rep. TR-CS-07-03, The Australian National University, 2007.

[30] Christen, P. Automatic record linkage using seeded nearest neighbour and support vector machine classification. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining* (New York, NY, USA, 2008), KDD '08, ACM, pp. 151–159.

[31] Christen, P., and Pudjijono, A. Accurate synthetic generation of realistic personal information. In *Proceedings of the* 13th *Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining* (Bangkok, Thailand, 2009), pp. 507–514.

[32] Churches, T., and Christen, P. Some methods for blindfolded record linkage. *BMC Medical Informatics and Decision Making 4*, 1 (2004), 9.

[33] Clark, D. E. Practical introduction to record linkage for injury research. *Injury Prevention 10*, 3 (2004), 186–191.

[34] Clifton, C., Kantarcioglu, M., Doan, A., Schadow, G., Vaidya, J., Elmagarmid, A., and Suciu, D. Privacy-preserving data integration and sharing. In *Proceedings of the 9th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery* (2004), pp. 19–26.

[35] Cohen, W. W., Ravikumar, P., and Fienberg, S. E. A comparison of string distance metrics for name-matching tasks. In *Proceedings of IJCAI-03 Workshop on Information Integration* (Acapulco, Mexico, August 2003), pp. 73–78.

[36] Cohen, W. W., and Richman, J. Learning to match and cluster large high-dimensional data sets for data integration. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining* (New York, NY, USA, 2002), KDD '02, ACM, pp. 475–480.

[37] Committee on Technical and Privacy Dimensions of Information for Terrorism Prevention and Other National Goals, National Research Council. *Protecting Individual Privacy in the Struggle Against Terrorists.* The National Academies Press, Washington, DC, 2008.

[38] Dalenius, T. Finding a needle in a haystack of identifying anonymous census records. *Journal of Official Statistics 2*, 3 (1986), 329 – 336.

[39] Dong, X., Halevy, A., and Madhavan, J. Reference reconciliation in complex information spaces. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data* (New York, NY, USA, 2005), SIGMOD '05, ACM, pp. 85–96.

[40] Du, W., and Atallah, M. J. Protocols for secure remote database access with approximate matching. Tech. Rep. CERIAS TR 2000-15, CERIAS, Purdue University, 2001.

[41] Durham, E., Xue, Y., Kantarcioglu, M., and Malin, B. Private medical record linkage with approximate matching. In *Proceedings of the American Medical Informatics Association Annual Symposium* (2010), pp. 182 – 186.

[42] Durham, E., Xue, Y., Kantarcioglu, M., and Malin, B. Quantifying the correctness, computational complexity, and security of privacy-preserving string comparators for record linkage. *Information Fusion* (2012).

[43] Elfeky, M., Verykios, V., and Elmagarmid, A. TAILOR: a record linkage toolbox. In *Data Engineering, 2002. Proceedings. 18th International Conference on* (2002), pp. 17 –28.

[44] Elmagarmid, A. K., Ipeirotis, P. G., and Verykios, V. S. Duplicate record detection: A survey. *IEEE Trans. on Knowl. and Data Eng. 19* (January 2007), 1–16.

[45] Eycken, E. V., Haustermans, K., Buntinx, F., Ceuppens, A., Weyler, J., Wauters, E., Oyen, H. V., Schaever, M. D., den Berge, D. V., and Haelterman, M. Evaluation of the encryption procedure and record linkage in the belgian national cancer registry. *Archives of Public Health 58* (2000), 281–294.

[46] Fellegi, I., and Sunter, A. A theory for record linkage. *Journal of the American Statistical Society* (1969), 1183–1210.

[47] Fulton, S. EU seeks privacy enforcement rights in US courts through diplomatic agreement. *International Business Times* (May 27 2010).

[48] Gomatam, S., Carter, R., Ariet, M., and Mitchell, G. An empirical comparison of record linkage procedures. *Statistics in Medicine 21*, 10 (2002), 1485–1496.

[49] Grannis, S. J., Overhage, J. M., Hui, S., and McDonald, C. J. Analysis of a probabilistic record linkage technique without human review. In *Proceedings of the American Medical Informatics Association Annual Symposium* (Washington, DC, 2003), pp. 259–263.

[50] Grannis, S. J., Overhage, J. M., and McDonald, C. Real world performance of approximate string comparators for use in patient matching. In *Proceedings of Medinfo* (San Francisco, CA, 2004), pp. 43–47.

[51] Grannis, S. J., Overhage, J. M., and McDonald, C. J. Analysis of identifier performance using a deterministic linkage algorithm. In *Proceedings of the American Medical Informatics Association Annual Symposium* (San Antonio, TX, 2002), pp. 305–309.

[52] Gusfield, D., and Irving, R. W. *The stable marriage problem: structure and algorithms.* MIT Press, Cambridge, MA, USA, 1989.

[53] Hall, R., and Fienberg, S. Privacy-preserving record linkage. In *Privacy in Statistical Databases*, J. Domingo-Ferrer and E. Magkos, Eds., vol. 6344 of *Lecture Notes in Computer Science.* Springer Berlin / Heidelberg, 2011, pp. 269–283.

[54] Hernández, M. A., and Stolfo, S. J. The merge/purge problem for large databases. In *Proceedings of the 1995 ACM SIGMOD international conference on Management of data* (New York, NY, USA, 1995), SIGMOD '95, ACM, pp. 127–138.

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

[56] Herzog, T., Scheueren, F., and Winkler, W. *Data quality and record linkage techniques.* Springer, 2007.

[57] Hieb, B. R. A proposal for a national health care identifier. In *Proceedings of the American Medical Informatics Association Annual Symposium* (1994), pp. 469 – 472.

[58] Hjaltason, G., and Samet, H. Properties of embedding methods for similarity searching in metric spaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence 25* (2003), 530–549.

[59] Hjaltason, G. R., and Samet, H. Incremental distance join algorithms for spatial databases. *SIGMOD Rec. 27* (June 1998), 237–248.

[60] Hylton, J. A. Identifying and merging related bibliographic records. Master's thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, February 1996.

[61] Inan, A., Kantarcioglu, M., Bertino, E., and Scannapieco, M. A hybrid approach to private record linkage. In *Proceedings of the IEEE International Conference on Data Engineering* (Cancun, Mexico, 2008), pp. 496–505.

[62] Inan, A., Kantarcioglu, M., Ghinita, G., and Bertino, E. Private record matching using differential privacy. In *Proceedings of the 13th International Conference on Extending Database Technology* (New York, NY, USA, 2010), EDBT '10, ACM, pp. 123–134.

[63] Inan, A., Kaya, S., Saygin, Y., Savas, E., Hintoglu, A., and Levi, A. Privacy preserving clustering on horizontally partitioned data. *Data and Knowledge Engineering 63* (2007), 646–666.

[64] Indyk, P., and Motwani, R. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing* (1998), pp. 604–613.

[65] Jain, N. Using Bloom filters to refine web search results. In *In Proceedings of the* 7th *International Workshop on Web Databases* (Baltimore, MD, 2005), pp. 25–30.

[66] Jaro, M. A. *UNIMATCH: a record linkage system.* U.S. Census Bureau, Washington, DC, 1978.

[67] Jin, L., Li, C., and Mehrotra, S. Efficient record linkage in large data sets. In *Proceedings of the Eighth International Conference on Database Systems for Advanced Applications* (March 2003), pp. 137 – 146.

[68] Jonas, J., Pattak, P. B., and Litchko, J. P. Using advanced information technology to combat insider threats. *Journal of Organizational Excellence 20*, 4 (2001), 19–27.

[69] Kantarcioglu, M., Inan, A., Jiang, W., and Malin, B. Formal anonymity models for efficient privacy-preserving joins. *Data Knowledge Engineering 68*, 11 (2009), 1206–1223.

[70] Karakasidis, A., and Verykios, V. S. Privacy preserving record linkage using phonetic codes. In *Proceedings of the* 4th *Balkan Conference in Informatics* (Thessaloniki, Greece, 2009), pp. 101–106.

[71] Kaya, S. V., Pedersen, T. B., Savas, E., and Saygin, Y. *Efficient Privacy Preserving Distributed Clustering Based on Secret Sharing*, vol. 4819 of *Lecture Notes in Computer Science.* Springer, 2007, pp. 280–291.

[72] Kim, H.-s., and Lee, D. HARRA: fast iterative hashed record linkage for large-scale data collections. In *Proceedings of the 13th International Conference on Extending Database Technology* (2010), pp. 525–536.

[73] Krouse, W., and Elias, B. Terrorist watchlist checks and air passenger prescreening. Tech. Rep. RL33645, Congressional Research Service, December 2009.

[74] Kulis, B., and Grauman, K. Kernelized locality-sensitive hashing for scalable image search. In *IEEE International Conference on Computer Vision (ICCV* (2009).

[75] Kushner, D. Vegas 911. *IEEE Spectrum 43* (2006), 44–49.

[76] Kuzu, M., Kantarcioglu, M., Durham, E., and Malin, B. A constraint satisfaction cryptanalysis of Bloom filters in private record linkage. In *Proceedings of the 11th international conference on Privacy enhancing technologies* (2011), pp. 226–245.

[77] Lehti, P., and Fankhauser, P. Unsupervised duplicate detection using sample non-duplicates. In *Journal on Data Semantics VII*, S. Spaccapietra, Ed., vol. 4244 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2006, pp. 136–164.

[78] Lenz, R. Measuring the disclosure protection of micro aggregated business microdata. an analysis taking as an example the german structure of costs survey. *Journal of Official Statistics 22*, 4 (2006), 681–710.

[79] Levenshtein, V. I. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady 10* (Feb. 1966), 707–710.

[80] Li, T. A general model for clustering binary data. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining* (New York, NY, USA, 2005), KDD '05, ACM, pp. 188–197.

[81] Liu, H., and Motoda, H. *Feature Selection for Knowledge Discovery and Data Mining*. Kluwer Academic Publishers, Norwell, MA, USA, 1998.

[82] MacQueen, J. B. Some methods for classification and analysis of multivariate observations. In *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability* (1967), L. M. L. Cam and J. Neyman, Eds., vol. 1, University of California Press, pp. 281–297.

[83] Malek, B., and Miri, A. Secure dot-product protocol using trace functions. In *Information Theory, 2006 IEEE International Symposium on* (july 2006), pp. 927 –931.

[84] Malin, B., and Sweeney, L. How (not) to protect genomic data privacy in a distributed network: using trail re-identification to evaluate and design anonymity protection systems. *Journal of Biomedical Informatics 37* (2004), 179–192.

[85] Marzal, A., and Vidal, E. Computation of normalized edit distance and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence 15* (1993), 926–932.

[86] McCallum, A., and Ungar, L. Efficient clustering of high-dimensional data sets with application to reference matching. In *Knowledge Discovery and Data Mining* (2000), pp. 169–178.

[87] Michelson, M., and Knoblock, C. A. Learning blocking schemes for record linkage. In *Proceedings of the 21st national conference on Artificial intelligence - Volume 1* (2006), AAAI Press, pp. 440–445.

[88] Monge, A., and Elkan, C. An efficient domain-independent algorithm for detecting approximately duplicate database records, 1997.

[89] Newman, T. B., and Brown, A. N. Use of commercial record linkage software and vital statistics to identify patient deaths. *Journal of the American Medical Informatics Association 4*, 3 (1997), 233–237.

[90] Ordonez, C. Clustering binary data streams with k-means. In *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery* (New York, NY, USA, 2003), DMKD '03, ACM, pp. 12–19.

[91] Paillier, P. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of EUROCRYPT* (Prague, Czech Republic, 1999), pp. 223–238.

[92] Pang, C., Gu, L., Hansen, D., and Maeder, A. Privacy-preserving fuzzy matching using a public reference table. In *Intelligent Patient Management*, vol. 189. Springer Berlin / Heidelberg, 2009, pp. 71 – 89.

[93] Porter, E. H., and Winkler, W. E. *Approximate String Comparison and its Effect on an Advanced Record Linkage System.* 1997.

[94] Quantin, C., Bouzelat, H., Allaert, F., Benhamiche, A., Faivre, J., and Dusserre, L. How to ensure data security of an epidemiological follow-up:quality assessment of an anonymous record linkage procedure. *International Journal of Medical Informatics 49*, 1 (1998), 117 – 122.

[95] Ravikumar, P., and Fienberg, S. E. A secure protocol for computing string distance metrics. In *In Proceedings of the IEEE Workshop on Privacy and Security Aspects of Data Mining* (Brighton, England, 2004), pp. 40–46.

[96] Rivest, R. L., Shamir, A., and Adleman, L. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM 21* (February 1978), 120–126.

[97] Roozenburg, J. A literature survey on bloom filters. Unpublished research assignment, 2005.

[98] Safran, C., Bloomrosen, M., Hammond, W. E., Labkoff, S., Markel-Fox, S., Tang, P. C., and Detmer, D. E. Toward a national framework for the secondary use of health data: An American Medical Informatics Association white paper. *Journal of the American Medical Informatics Association 14*, 1 (2007), 1–9.

[99] Scannapieco, M., Figotin, I., Bertino, E., and Elmagarmid, A. Privacy preserving schema and data matching. In *Proceedings of the ACM SIGMOD International Conference on Management of Data* (Beijing, China, 2007), pp. 653–664.

[100] Schneier, B. *Applied Cyptogography: Protocols, Algorithms and Source Code in C.* Wiley, 1996.

[101] Schnell, R., Bachteler, T., and Reiher, J. Privacy-preserving record linkage using bloom filters. *BMC Medical Informatics and Decision Making 9*, 1 (2009), 41.

[102] Slaney, M., and Casey, M. Locality-sensitive hashing for finding nearest neighbors. *IEEE Signal Processing Magazine 25* (Mar. 2008), 128–131.

[103] Song, D. X., Wagner, D., and Perrig, A. Practical techniques for searches on encrypted data. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy* (Oakland, CA, 2000), b, p. 44.

[104] Stallings, W. *Cryptography and Network Security: Principles and Practice.* Pearson Education, 2002.

[105] Stanley, F. J., Croft, M. L., Gibbins, J., and Read, A. W. A population database for maternal and child health research in Western Australia using record linkage. *Paediatric and Perinatal Epidemiology 8*, 4 (1994), 433–447.

[106] Stead, W. W., Kelly, B. J., and Kolodner, R. M. Achievable steps toward building a national health information infrastructure in the United States. *Journal of the American Medical Informatics Association 12*, 2 (2005), 113–120.

[107] Tejada, S., Knoblock, C. A., and Minton, S. Learning domain-independent string transformation weights for high accuracy object identification. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining* (New York, NY, USA, 2002), KDD '02, ACM, pp. 350–359.

[108] Torra, V., and Domingo-Ferrer, J. Record linkage methods for multidatabase data mining. In *Information fusion in data mining*. Springer, 2003, pp. 101–132.

[109] Trepetin, S. Privacy-preserving string comparisons in record linkage systems: A review. *Information Security Journal: A Global Perspective 17*, 5&6 (2008), 253–266.

[110] Tromp, M., Reitsma, J., Ravelli, A., Meray, N., and Bonsel, G. Record linkage: making the most out of errors in linking variables. In *Proceedings of the American Medical Informatics Association Annual Symposium* (2006), pp. 779–783.

[111] U.S. Census Bureau, Population Division. U.S. Census Name Files. `http://www.census.gov/genealogy/names/names_files.html`, Last accessed June 2, 2010.

[112] U.S. Department of Health & Human Services. HIPAA Privacy Rule, Last accessed December 30, 2011.

[113] U.S. General Accounting Office. Record linkage and privacy: issues in creating new federal research and statistical information. Tech. Rep. GAO-01-126SP, U.S. General Accounting Office, 2001.

[114] U.S. National Institutes of Health. Final NIH statement on sharing research data, NOT-OD-03-032, 2003.

[115] Verykios, V. S., Karakasidis, A., and Mitrogiannis, V. K. Privacy preserving record linkage approaches. *International Journal of Data Mining and Modelling and Management 1* (May 26 2009), 206–221.

[116] Weber, S. C., Lowe, H., Das, A., and Ferris, T. A simple heuristic for blindfolded record linkage. *Journal of the American Medical Informatics Association* (2012).

[117] Wetzel, T. G. New Sentinel system aims to improve patient safety in real time. *Hospitals & Health Networks 85* (2011), 12.

[118] Winkler, W. E. Using the EM algorithm for weight computation in the Fellegi-Sunter model of record linkage, Research Report RR00/05, U.S. Census Bureau, 2000.

[119] Winkler, W. E. Methods for record linkage and bayesian networks. Tech. rep., Series RRS2002/05, U.S. Bureau of the Census, 2002.

[120] Winkler, W. E. Overview of record linkage and current research directions. Tech. rep., BUREAU OF THE CENSUS, 2006.

[121] Yancey, W. E. Bigmatch: A program for large-scale record linkage. Tech. rep., U.S. Census Bureau, 2002.

[122] Yasnoff, W. A., Humphreys, B. L., Overhage, J. M., Detmer, D. E., Brennan, P. F., Morris, R. W., Middleton, B., Bates, D. W., and Fanning, J. P. A consensus action agenda for achieving the national health information infrastructure. *Journal of the American Medical Informatics Association 11*, 4 (2004), 332–338.